

# Machine Learning (CE 40477)

Fall 2024

Ali Sharifi-Zarchi

CE Department  
Sharif University of Technology

December 9, 2025



## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

## 4 Data Preprocessing

## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

## 1 Why Deep Networks?

## ② AlexNet

3 ResNet

## 4 Data Preprocessing

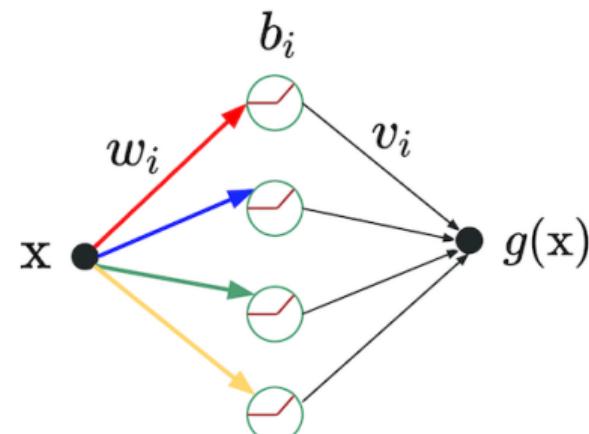
## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

## Why Deep Networks? (Deep Expressivity)

- While the universal approximation theorem states that a single-hidden-layer neural network can approximate any function to a certain level of accuracy, it does not specify how many hidden neurons are needed.
    - It states there exists an integer  $N$  and a set of parameters  $\{v_i, w_i, b_i\}_{i=1}^N$
    - **Depth Separation Theorem:** For any integer  $k > 0$ , there exist neural networks with  $\Theta(k^3)$  layers,  $\Theta(1)$  nodes per layer, and  $\Theta(1)$  distinct parameters which can't be approximated by networks with  $O(k)$  layers unless they are exponentially large (they must possess  $\Omega(2^k)$  nodes/neurons).



Telgarsky. Benefits of depth in neural networks. PMLR (2016).

## Intuition

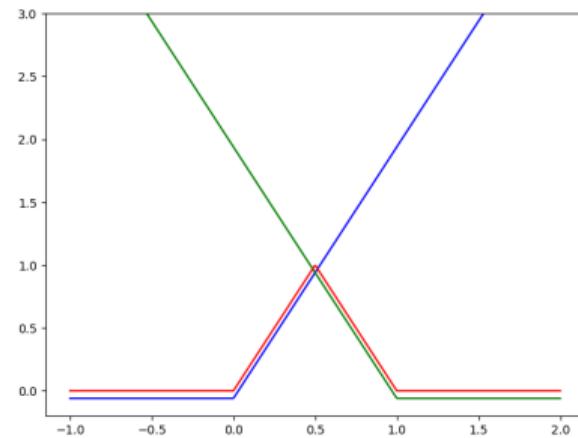
- Let's consider ReLU networks. Let  $m(x)$  be the piecewise linear function as below:

$$m(x) = \begin{cases} 2x & 0 \leq x \leq 0.5 \\ 2 - 2x & 0.5 \leq x \leq 1 \\ 0 & o.w. \end{cases}$$

- This can be computed exactly by a two- or three-layer network (depending on your notational preferences), by the expression:

$$\sigma(2\sigma(x) - 4\sigma(x - 0.5))$$

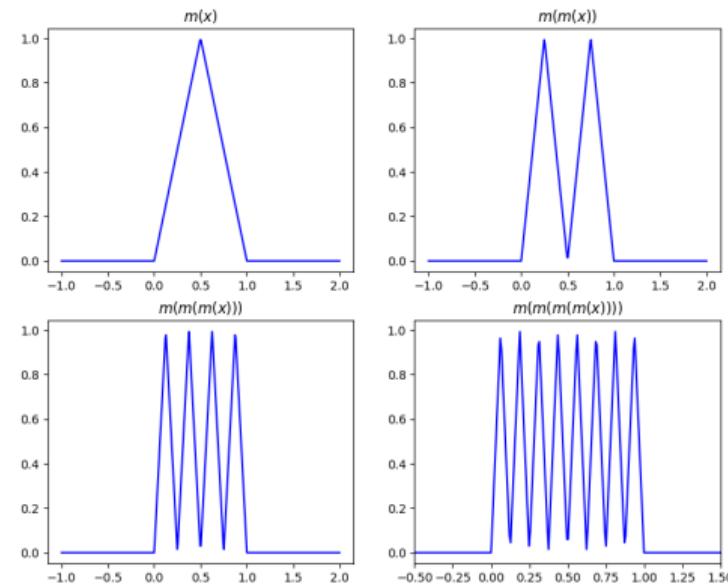
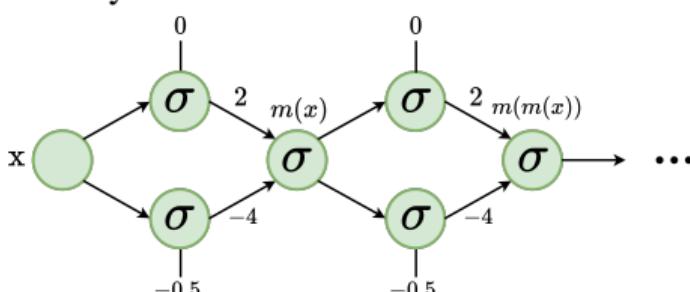
- Figure on the right shows it in terms of ReLU activation functions



Function  $m$  is slightly offset for visual clarity

## Intuition

- What do iterates of  $m(x)$  look like?
  - So  $m^{(n)}(x)$  would have  $2^n$  “teeth”
  - We can represent this function with  $3n + 1$  nodes
  - A shallow network requires many nodes to achieve this
  - Depth increases the number of oscillations multiplicatively whereas width can only do so additively.



## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

## 4 Data Preprocessing

## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

AlexNet [Krizhevsky, et al. NIPS (2012)]

## Architecture:

[ $227 \times 227 \times 3$ ] INPUT

[ $55 \times 55 \times 96$ ] CONV1 + ReLU: 96  $11 \times 11$  filters at stride 4, pad 0

[ $27 \times 27 \times 96$ ] **MAX POOL1**: MAX POOL1;  $3 \times 3$  filters at stride 2

[ $27 \times 27 \times 96$ ] **NORM1**: Normalization layer

[ $27 \times 27 \times 256$ ] **CONV2 + ReLU**: 256  $5 \times 5$  filters at stride 1, pad 2

[ $13 \times 13 \times 256$ ] MAX POOL2:  $3 \times 3$  filters at stride 2

[ $13 \times 13 \times 256$ ] NORM2: Normalization layer

[ $13 \times 13 \times 384$ ] CONV3 + ReLU: 384  $3 \times 3$  filters at stride 1, pad 1

[ $13 \times 13 \times 384$ ] CONV4 + ReLU: 384  $3 \times 3$  filters at stride 1, pad 1

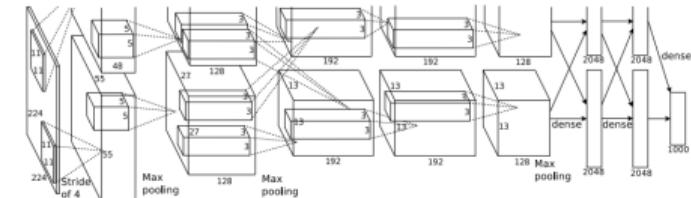
[ $13 \times 13 \times 256$ ] CONV5 + ReLU: 256  $3 \times 3$  filters at stride 1, pad 1

[ $6 \times 6 \times 256$ ] MAX POOL3:  $3 \times 3$  filters at stride 2

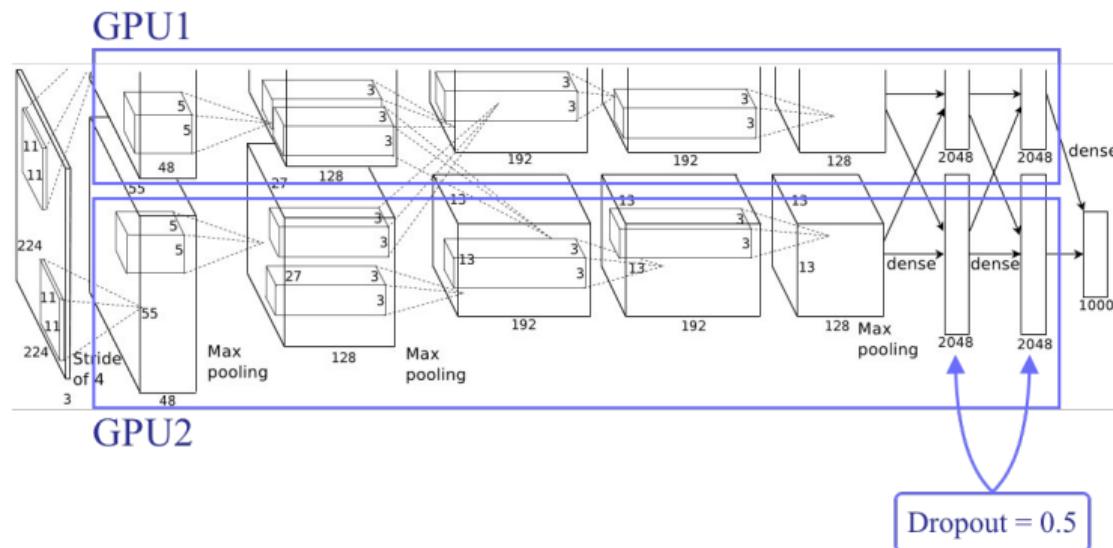
[4096] Fully Connected 6 + ReLU 4096 neuron

[4096] Fully Connected 7 + ReLU 4096 neuron

[1000] Fully Connected 8 1000 neurons (class scores)



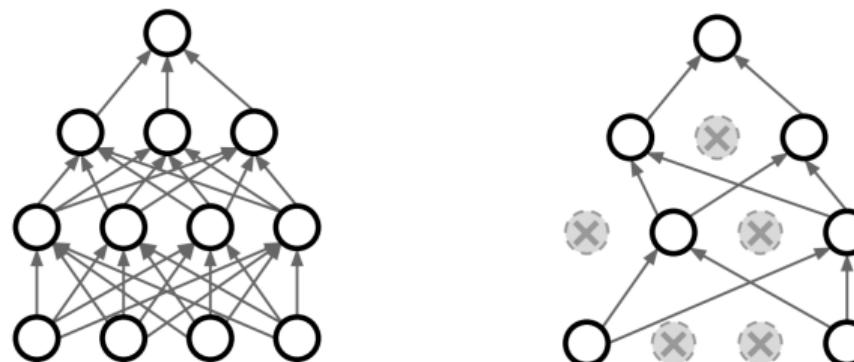
## AlexNet At A Glance



- Training: 6 days on 2 NVIDIA GTX 580 GPUs (3 GB RAM)
  - 8 layers / 60M parameters and 650K neurons
  - Testing: Multi-crop
    - Classify different shifts of the image and vote over the lot!
    - Test-time data augmentation

## Dropout

- In each forward pass, randomly deactivate some neurons.
    - Probability of dropping is a hyperparameter  $\Rightarrow 0.5$  is common
    - Gradients are computed only for the weights and biases between active nodes
    - For the remaining, the gradient is just 0.



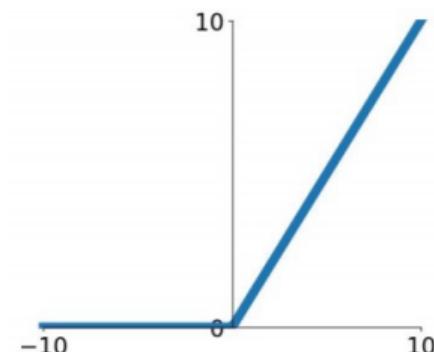
Srivastava et al., JMLR 2014

- Thus, a smaller network is trained for each sample  
→ Smaller network provides a regularization effect.

ReLU

## ReLU

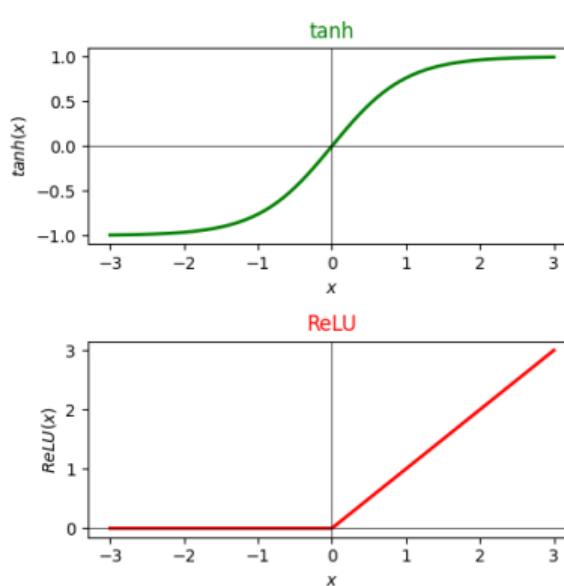
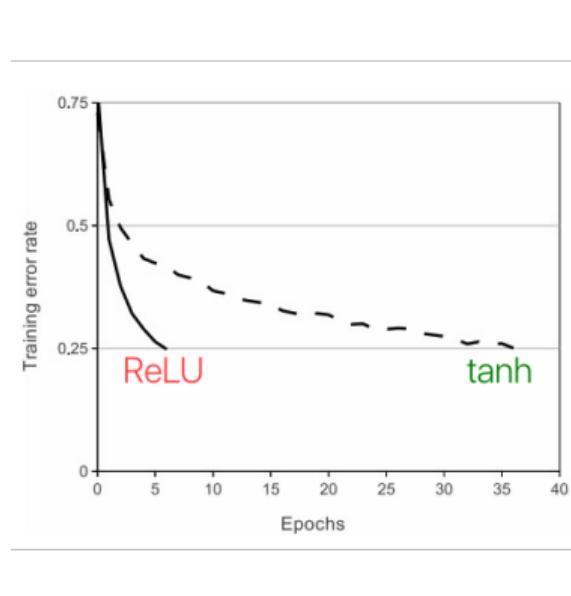
- Does not saturate (in the positive region)
  - Very computationally efficient
  - Converges much faster than sigmoid/tanh in practice (e.g. 6x)
  - Actually more biologically plausible than sigmoid
  - **Problems:**
    - Not zero-centered output
    - An annoyance:
      - ★ Hint: "what is the gradient when  $x < 0$ ?"



## ReLU

## ReLU

- ReLU trains faster than tanh.



Krizhevsky, et al., 2012

## Local Response Normalisation

$$b_{x,y}^i = \frac{a_{x,y}^i}{\left( k + \alpha \sum_{j=\max(0, i-\frac{n}{2})}^{\min(N-1, i+\frac{n}{2})} \left( a_{x,y}^j \right)^2 \right)^\beta}$$

- Enhance the generalization of the model
  - Encourages competition between adjacent features
    - $b_{x,y}^i$ : Normalized activation at spatial location  $(x, y)$  in the  $i$ -th channel.
    - $a_{x,y}^i$ : Original activation at the same location and channel.
    - $N$ : Total number of channels.
    - $n$ : Size of the local neighborhood (number of adjacent channels to consider for normalization).
    - $k, \alpha, \beta$ : Hyperparameters controlling the normalization.

## Learning Magic In Alexnet

- First use of ReLU
    - Made a large difference in convergence
  - Dropout probability 0.5 (in FC layers only)
  - Augmentations: translation, horizontal flip, altering the intensities of the RGB channels
  - SGD with momentum of 0.9 and a mini-batch size of 128
  - L2 weight decay 5e-4 ( $l = l_{data} + \lambda ||w||^2$ )
  - Learning rate: 0.01, decreased by 10 every time validation accuracy plateaus
  - Evaluation metric: Validation accuracy
  - **Final top-5 error: 18.2% with a single net, 15.4% using an ensemble of 7 networks**
    - Lowest prior error using conventional classifiers: > 25 %

## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

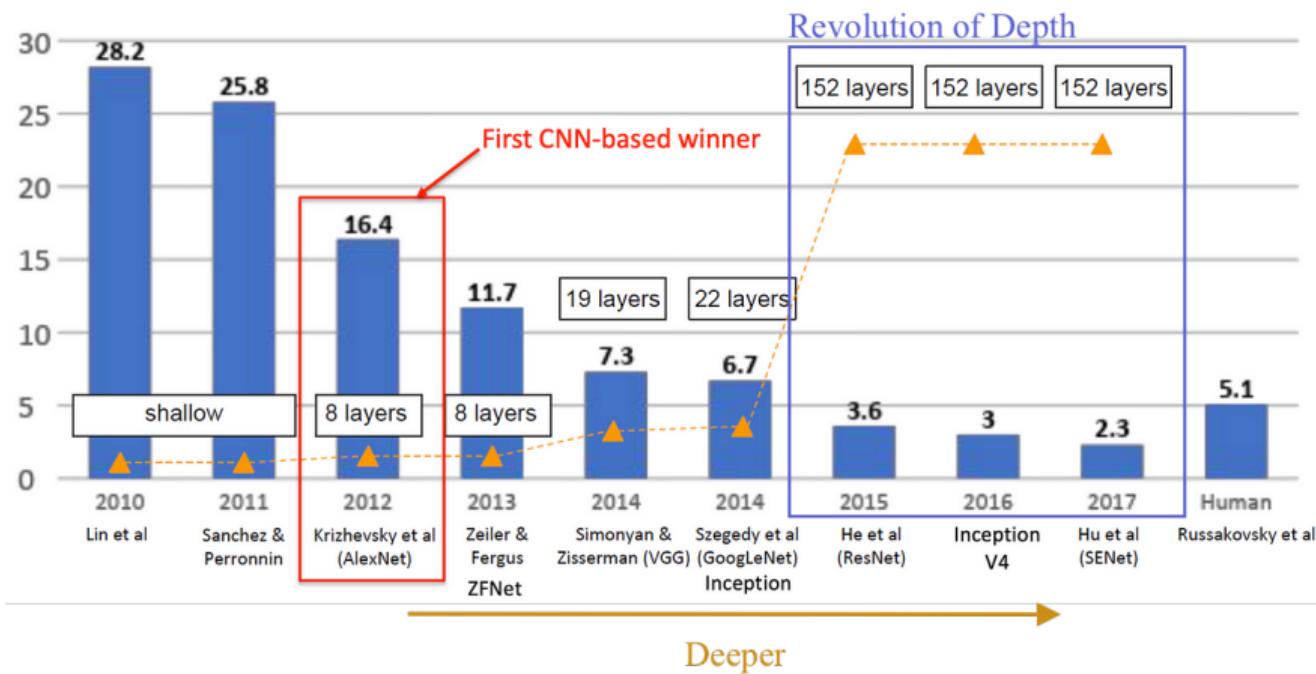
## 4 Data Preprocessing

## 5 Data Augmentation

## 6 Transfer Learning

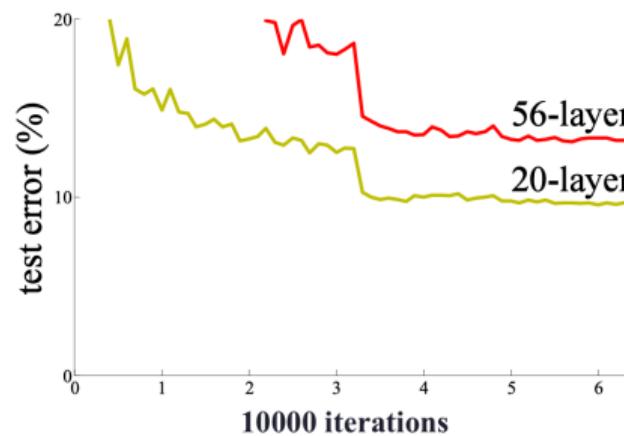
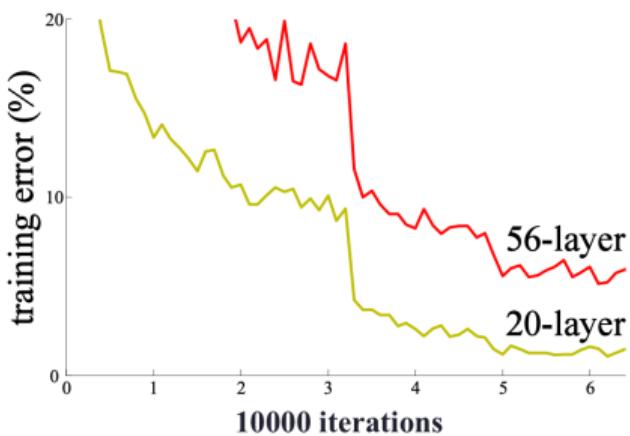
## 7 References

# Revolution Of Depth



## Challenge Of Making CNNs Deeper

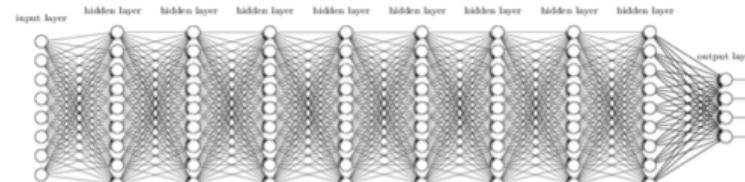
- What happens when we continue stacking deeper layers on a convolutional neural network?
  - **Question:** What is strange about these training and test curves?



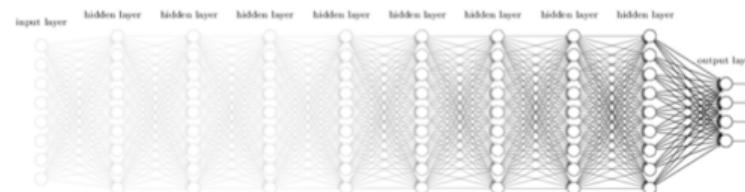
Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks

# Challenge Of Making CNNs Deeper

- **Fact:** Deeper models have more representation power (more parameters) than shallower models.
  - But **56-layer model performs worse** on both training and test error and it's not caused by overfitting!
  - **Hypothesis:** The issue is related to optimization, deeper models are harder to optimize.



Deep Neural Network



Vanishing Gradient

## Vanishing Gradient

- Consider a deep neural network without residual blocks. If we have the error signal  $\bar{h}^{(T)}$ , we then alternate between the following two backprop rules:

$$\bar{h}^{(t)} \equiv z^{(t+1)} w, \quad (1)$$

$$z^{(t)} = \bar{h}^{(t)} \phi'(z^{(t)}). \quad (2)$$

- If we iterate the rules, we get:

$$\bar{h}^{(1)} = w^{T-1} \phi'(z^{(2)}) \cdots \phi'(z^{(T)}) \bar{h}^{(T)} \quad (3)$$

$$= \frac{\partial h^{(T)}}{\partial h^{(1)}} \bar{h}^{(T)}. \quad (4)$$

## Vanishing Gradient

- Hence,  $\bar{h}^{(1)}$  is a linear function of  $\bar{h}^{(T)}$ ; the coefficient is the partial derivative  $\frac{\partial h^{(T)}}{\partial h^{(1)}}$ . If we make the simplifying assumption that the activation functions are linear, we get:

$$\frac{\partial h^{(T)}}{\partial h^{(1)}} = w^{T-1}, \quad (5)$$

- which can clearly explode or vanish unless  $w$  is very close to 1. For instance, if  $w = 1.1$  and  $T = 50$ , then  $\partial h^{(T)} / \partial h^{(1)} \approx 117.4$ , whereas if  $w = 0.9$  and  $T = 50$ , we get  $\partial h^{(T)} / \partial h^{(1)} = 0.00515$ .

## Jacobian

- More generally, in the multivariate case,

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}. \quad (6)$$

- This quantity is called the *Jacobian*. In the case of linear activation functions,

$$\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} = \mathbf{W},$$

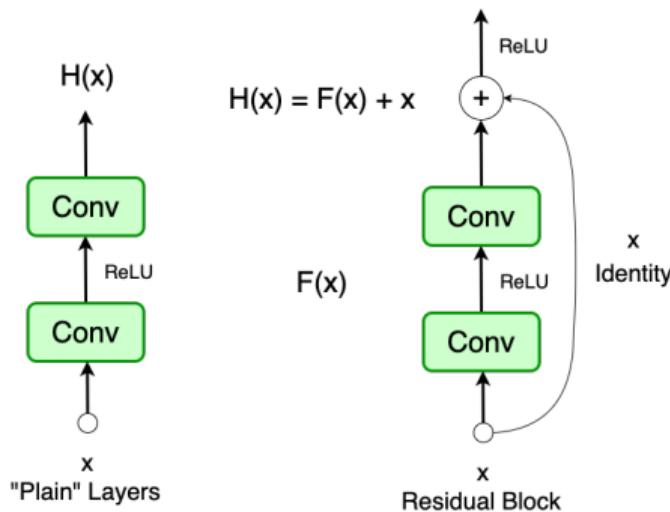
so

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \mathbf{W}^{T-1}. \quad (7)$$

- This will explode if the largest eigenvalue of  $\mathbf{W}$  is larger than 1, and vanish if the largest eigenvalue is smaller than 1.

## Residual Block

- **Solution:** Use network layers to fit a residual mapping rather than directly fitting the desired underlying mapping.
  - Identity mapping:  $H(\mathbf{x}) = \mathbf{x}$  if  $F(\mathbf{x}) = 0$



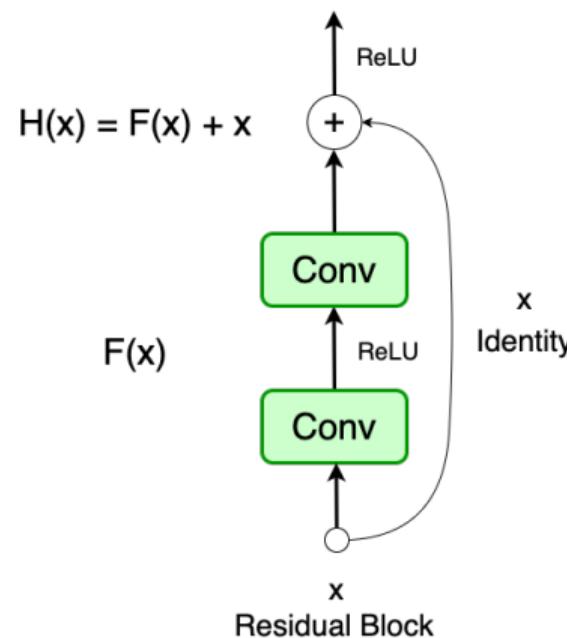
Use layers to fit residual  $F(x) = H(x) - x$  instead of  $H(x)$  directly

# Backpropagation In Residual Block

- How exactly does the skip connection prevent gradient vanishing?
- **During backpropagation, there are two pathways.**

$$H(x) = F(x) + x$$

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

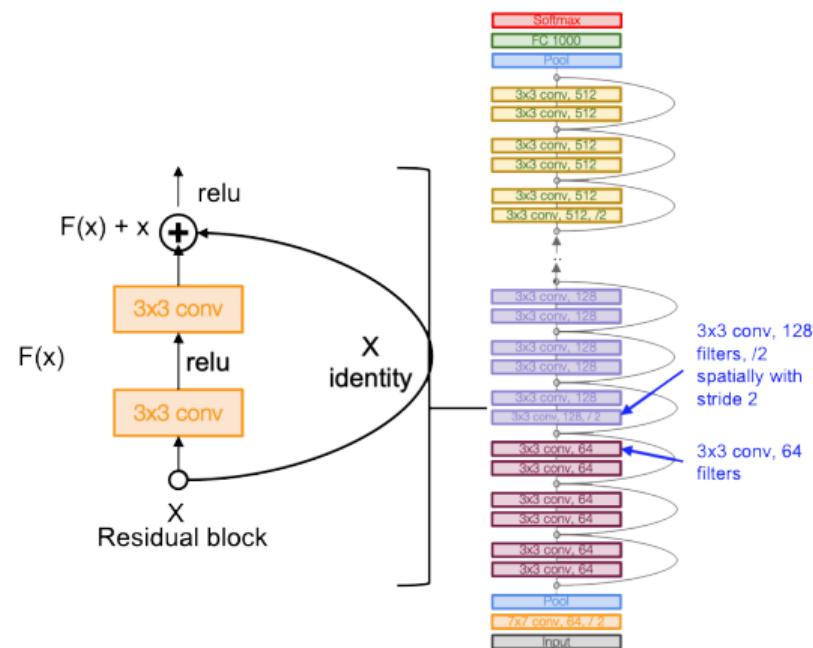


- If the gradient of Path 1 vanishes to 0, the gradient of Path 2 still remains non-zero (because this gradient does not encounter any weight layer)

# ResNet [He, et al., (2015)]

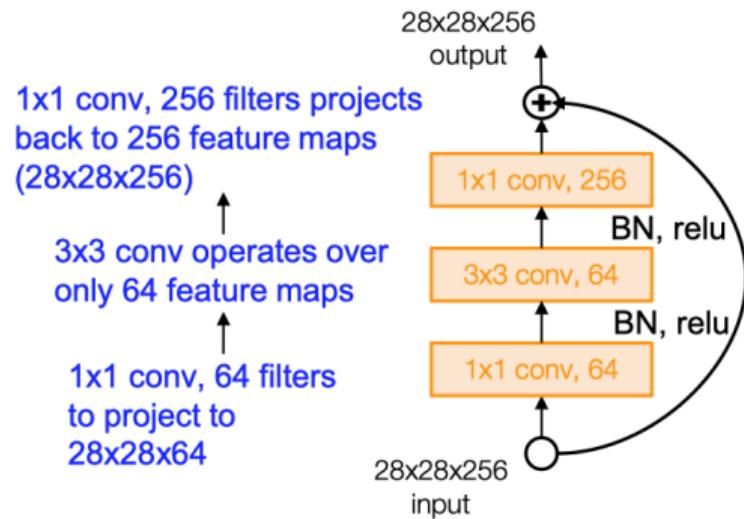
## Architecture:

- Stack of residual blocks
- Doubling the filter count while halving spatial dimensions (via stride-2) at regular intervals in the network Reduce the activation volume by half.
- One FC layer at the end to output classes
- Total depths of 18, 34, 50, 101, or 152 layers for ImageNet



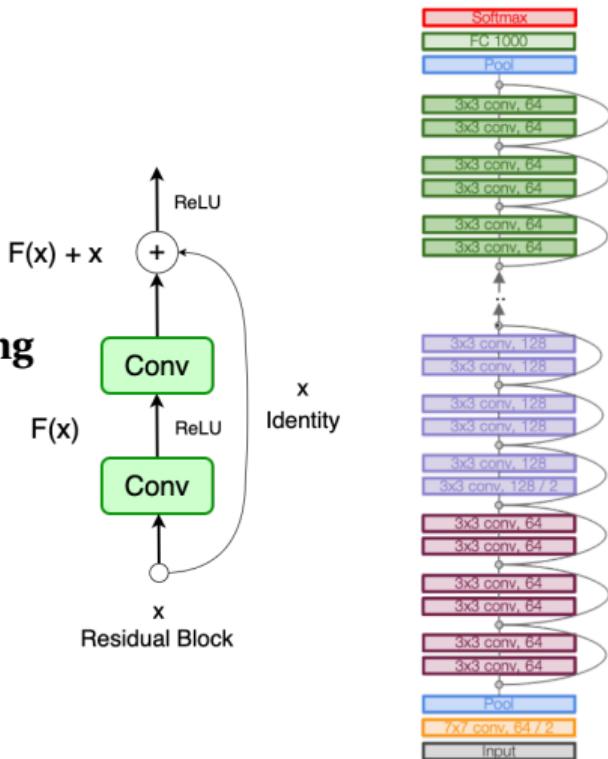
# Deeper Residual Module (Bottleneck)

- For deeper networks (ResNet- 50+), use “bottleneck” layer instead, to improve efficiency.
- Bottleneck uses 1x1 convolution to:
  - Reduce number of parameters
  - Enable interaction between channels, creating new feature representations at each spatial location
  - Allow more layers to be added without a significant increase in computational cost or overfitting risk.



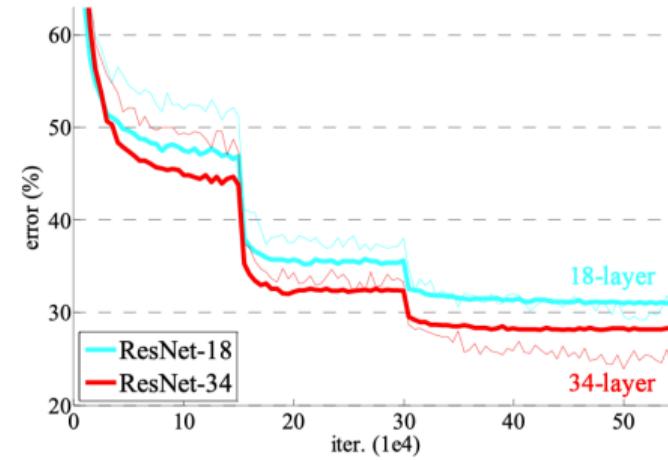
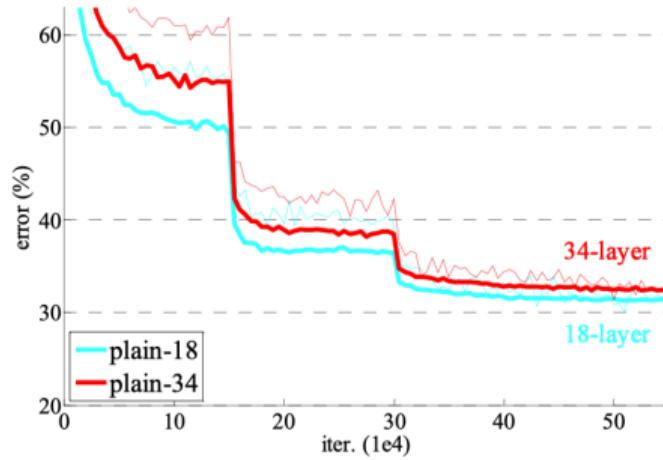
# ResNet At A Glance

- Very deep networks using residual connections
- ResNet **introduces** the concept of skip connections (or residual connections) that allow the gradient to be directly backpropagated to earlier layers.
- **Skip connections helps overcoming gradient vanishing** problem, where the accuracy saturates and then degrades rapidly as the network depth increases.
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57 % top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



## ResNet Experiments

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on CIFAR)
- Deeper networks now achieve lower training error as expected.



Thin curves represent training error, and bold curves represent validation error [He, et al. 2015]

# Training ResNet In Practice:

- Batch Normalization after every CONV layer
- Xavier 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

# Key Points

- **AlexNet** showed that you can use CNNs to train Computer Vision models.
- **ResNet** showed us how to train extremely deep networks.
  - Limited only by GPU & memory!
  - Showed diminishing returns as networks got bigger (increasing the depth or size of the network doesn't proportionally improve its performance)
- After ResNet: CNNs were better than the human metric and focus shifted to other topics:
  - ★ Efficient Networks: **MobileNet, ShuffleNet**
  - ★ **Neural Architecture Search** can now automate architecture design

## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

## 4 Data Preprocessing

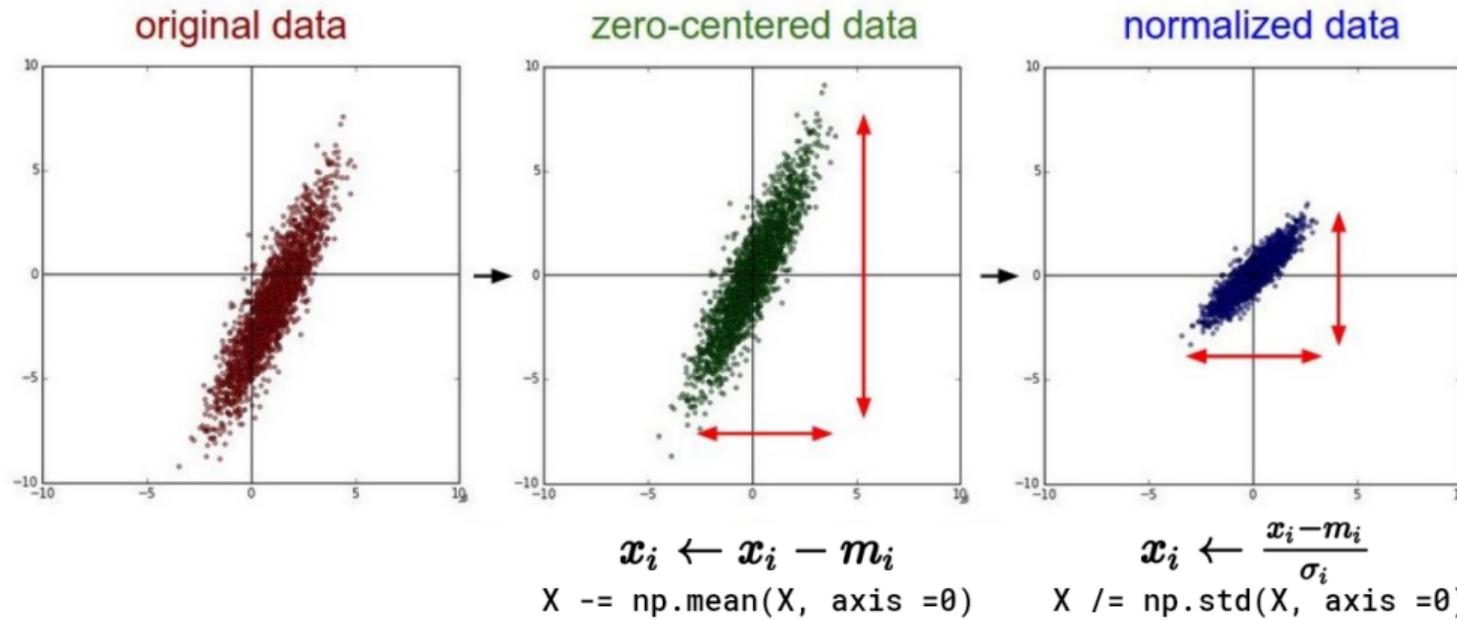
## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

## Normalizing The Data

- Assume  $X_{n \times d}$  is data matrix, with each sample is represented in a row
  - $i$  is the index of dimension ( $i = 1, \dots, d$ )

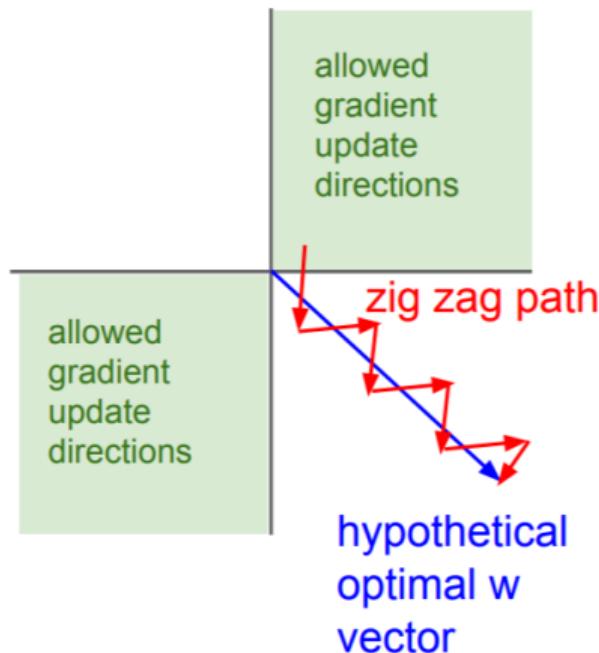


# TLDR: In Practice For Images: Center Only

- Example: consider CIFAR-10 example with  $[32, 32, 3]$  images
- 3 different approaches:
  - ① Subtract the mean image (e.g. AlexNet)
    - mean image =  $[32, 32, 3]$  array
  - ② Subtract per-channel mean (e.g. VGGNet)
    - mean along each channel = 3 numbers
  - ③ Subtract per-channel mean and divide by per-channel std (e.g. ResNet)

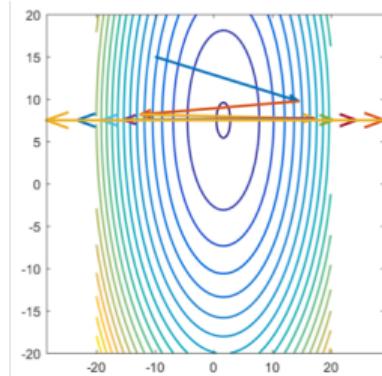
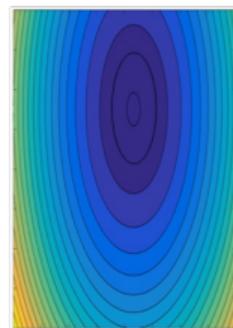
# Why Zero-Mean The Input?

- Reminder: sigmoid
- Consider what happens when the input to a neuron is always positive
- What can we say about the gradients on  $w$ ?
  - Always all positive or all negative
  - Leads to zig-zag dynamics in the gradient updates for the weights (in red) if the optimal  $w$  vector was the blue one
- This is also why zero-mean data is preferable.

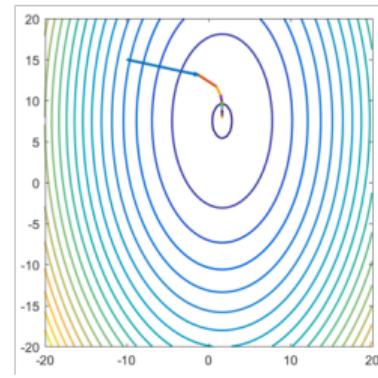
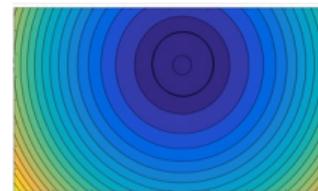


## Why Normalize The Input?

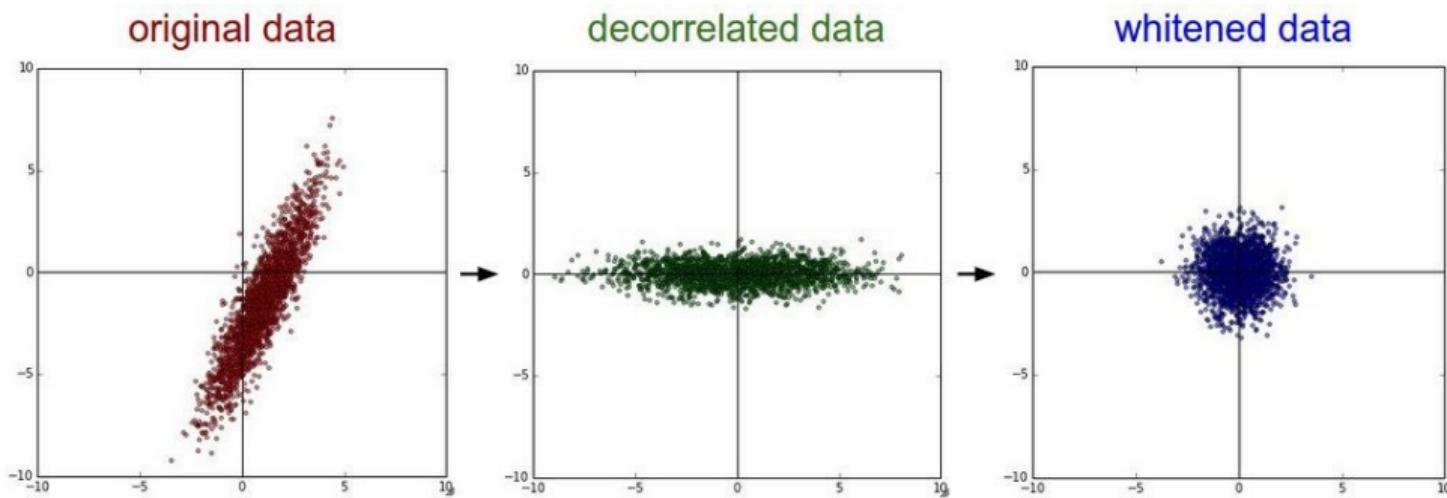
- Wide variation in the range of feature values
    - Leads to poor conditioning
  - Results in noisy movement of gradient



- Normalized data:
    - Enhances the geometry of the loss function
      - Mitigates issues of poor conditioning
    - Accelerates convergence during training



- While standardizing inputs is common practice, transformations like variance normalization, PCA, and whitening are used more selectively.



Data has diagonal covarianve matrix

Covarianve matrix  
is identity matrix

## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

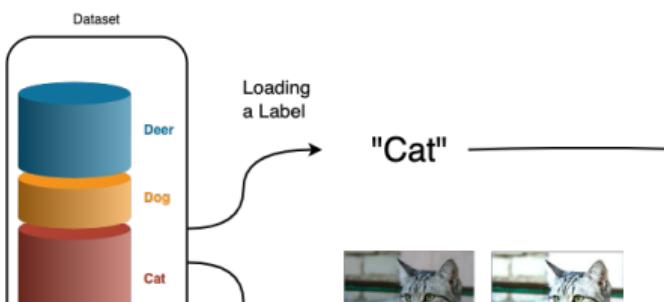
## 4 Data Preprocessing

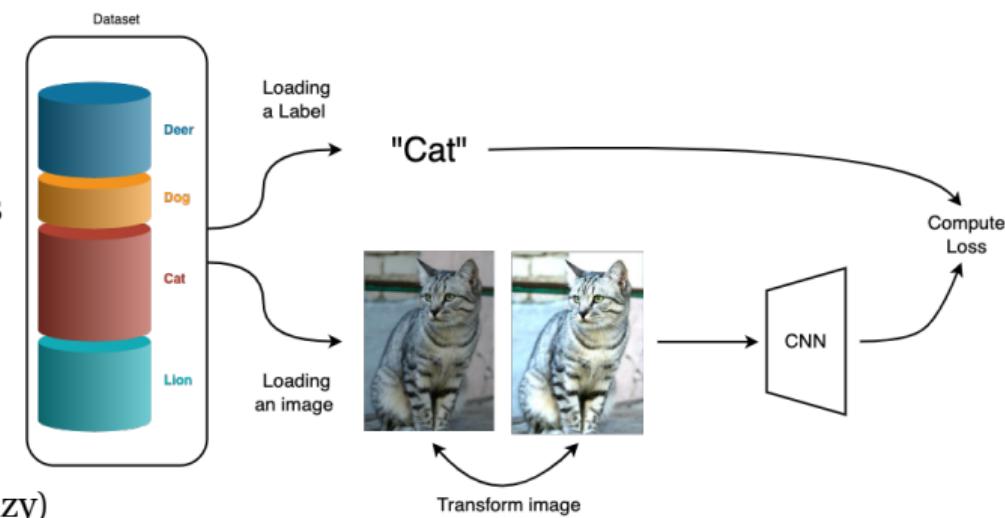
## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

## Motivation

- Getting more training data is often expensive
  - But, we can often generate additional training examples from existing datasets.
  - Transform each input data example in such a way that the label stays the same
  - Benefits:
    - Reduces the risk of overfitting
    - Improves generalization
    - Acts as a regularization
  - Examples of data augmentations
    - Translation
    - Flip (Horizontal/Vertical)
    - Rotation
    - Stretching
    - Shearing
    - Lens distortions, ... (going crazy)

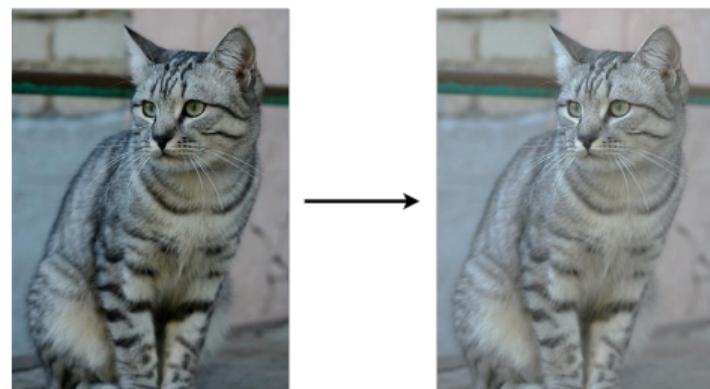


# Augmentations In Action

- Simple
  - Randomize contrast and brightness levels
- Complex
  - ① Apply PCA to all [R, G, B] pixels in training set
  - ② Sample a “color offset” along principal component directions
  - ③ Add offset to all pixels of a training image

(As seen in AlexNet, ResNet, etc)

Color Jitter



This image by [Nikita](#) is licensed under [CC-BY 2.0](#)

# Augmentations In Action

- **Training:** randomly sample crops and scales

ResNet:

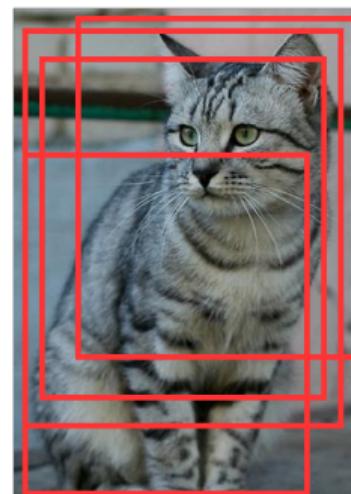
- ① Pick random L in range [256, 480]
- ② Resize training image, short side = L
- ③ Sample random 224 x 224 patch

- **Testing:** average a fixed set of crops

ResNet:

- ① Resize image at 5 scales: 224, 256, 384, 480, 640
- ② For each size, use 10 224 x 224 crops:  
4 corners + center + flips

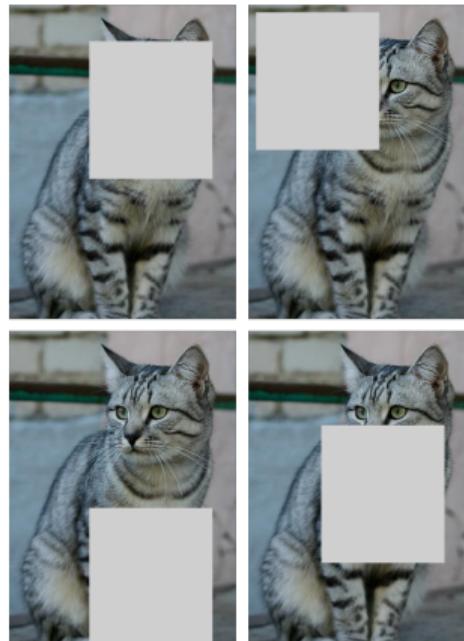
## Random Crops and Scales



# Augmentations In Action

- **Training:** Randomly set image regions to zero
- **Testing:** Use the full image
- Effective for small datasets like CIFAR, less common for large datasets like ImageNet

Cutout



DeVries and Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout", arXiv 2017

# Augmentation With Albumentations

```
1 import albumentations as A
```



- Basic pixel-Level Augmentations
  - A.RandomBrightnessContrast(), A.HueSaturationValue(), A.GaussianBlur()
- Image Distortion and Warping
  - A.ElasticTransform(), A.GridDistortion(), A.OpticalDistortion()
- And many more:
  - Advanced Pixel-Level Augmentations, Cropping and Padding, Geometric Transformations, Composite Augmentations, etc.
- Albumentations can even be used to **augment bounding boxes for object detection tasks** (check [this](#) out!)
- Interactive demo: <https://demo.albumentations.ai>

# Examples



A. GaussianBlur()

Blurs using Gaussian kernel

A. RandomGridShuffle()

Divides into a grid and shuffles the blocks

Image source: [freepik.com](https://www.freepik.com)

## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

## 4 Data Preprocessing

## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

## Motivation

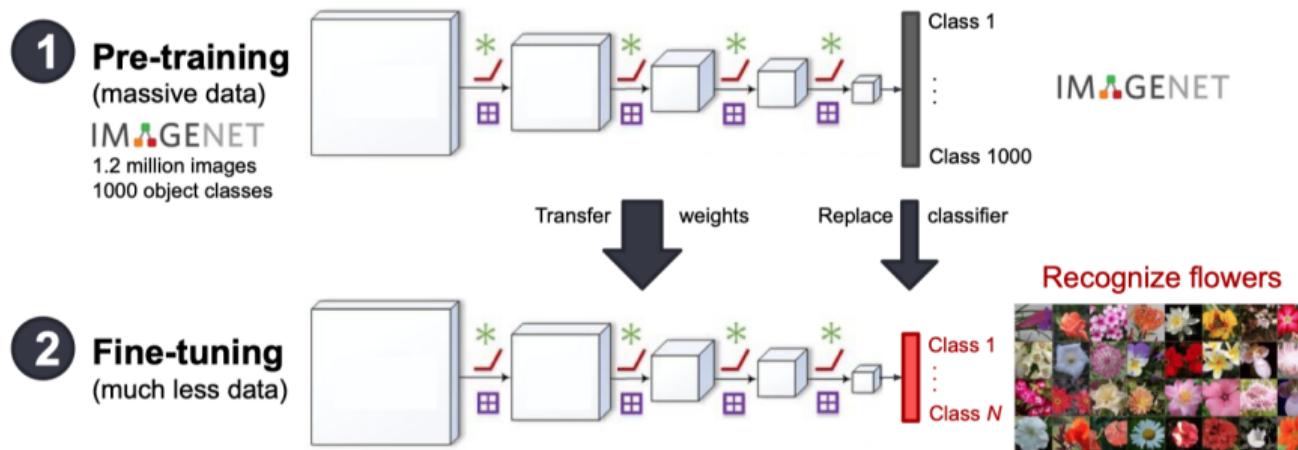
- You need a lot of data if you want to train/use CNNs
  - Suppose you want to build an app to recognize flowers...



<https://www.robots.ox.ac.uk/~vgg/data/flowers/102>

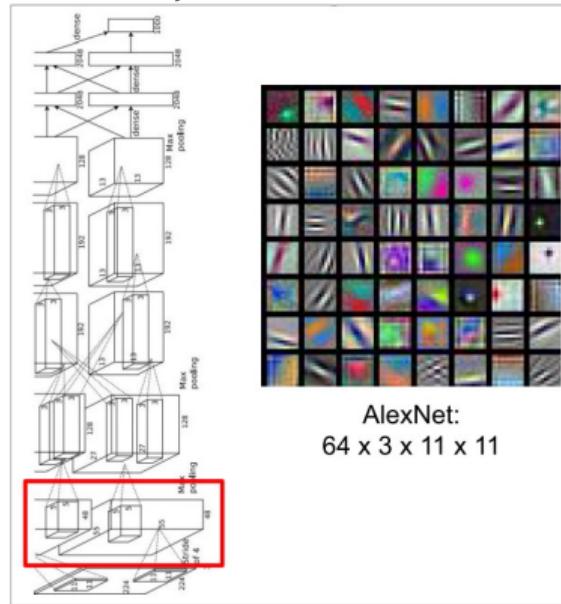
## Transfer Learning

- ① Pretrain on a large dataset (e.g. ImageNet)
  - ② Fine-tune it on a small target dataset (e.g. Flowers)

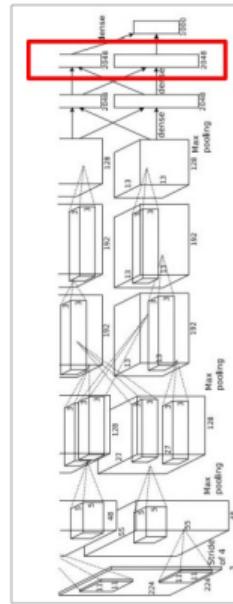


## Transfer Learning: The idea

Earlier Layers -> more local features



Final Layers -> more global features



Test image L2 Nearest neighbors in feature space

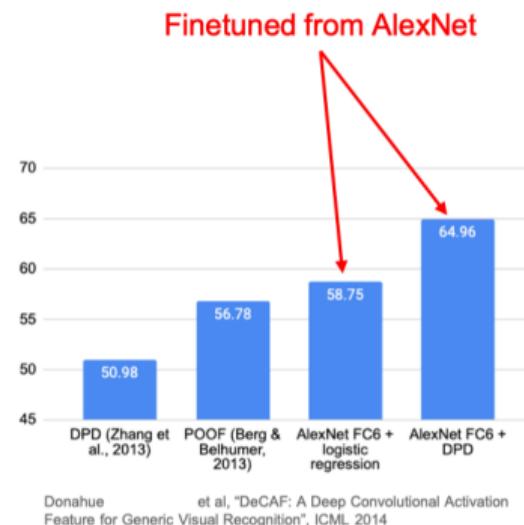
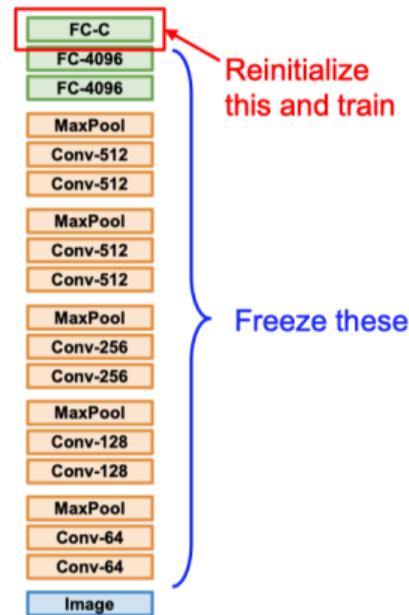


## Transfer Learning With CNNs

## 1. Train on Imagenet



## 2. Small Dataset (C classes)



Donahue et al, (ICML 2014), Razavian et al, (CVPR Workshops 2014)

## Transfer Learning With CNNs

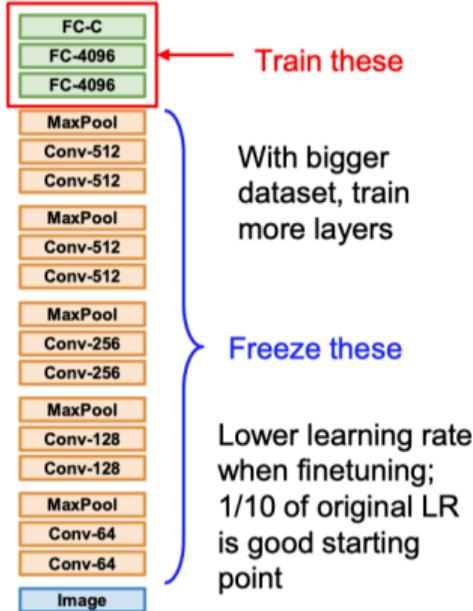
## 1. Train on Imagenet



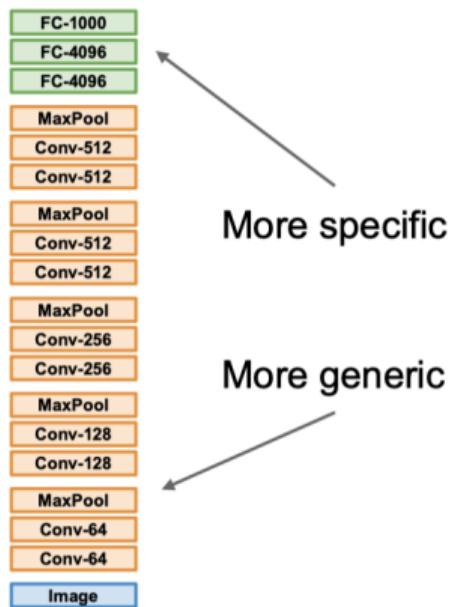
## 2. Small Dataset (C classes)



### 3. Bigger dataset



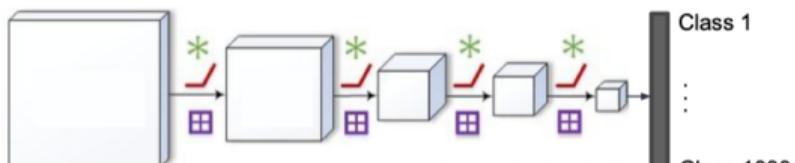
## Transfer Learning With CNNs



	<b>very similar dataset</b>	<b>very different dataset</b>
<b>very little data</b>	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
<b>quite a lot of data</b>	Finetune a few layers	Finetune a larger number of layers or start from scratch!

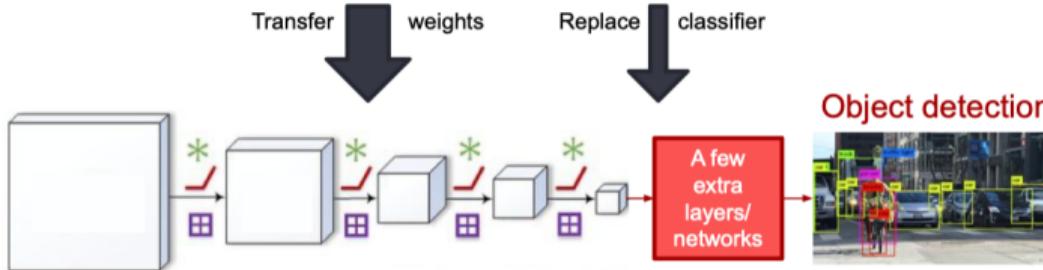
## Transfer Learning: Domain Shift

## 1 Pre-training (massive data)

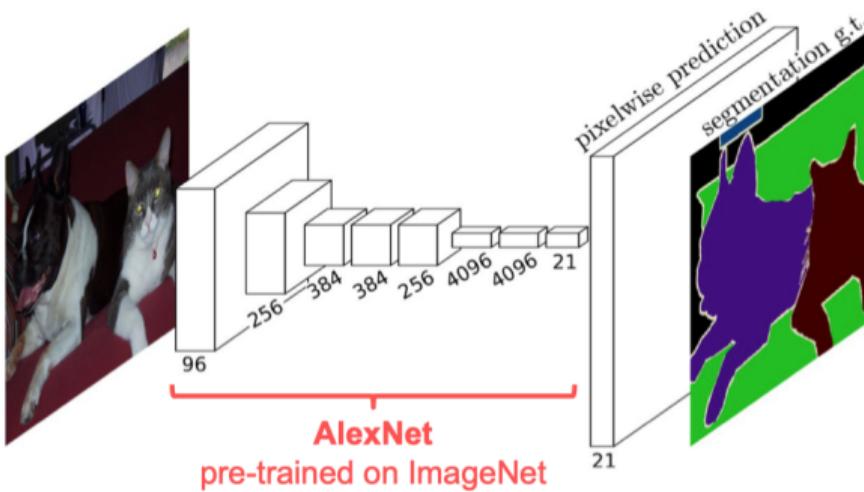


IMAGENET

## 2 Fine-tuning (much less data)



# Example: Semantic Segmentation



Long, Shelhamer, Darrell, CVPR 2015

# Transfer Learning With CNNs Is Pervasive...

Pose estimation

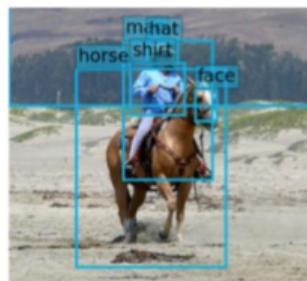


Image captioning



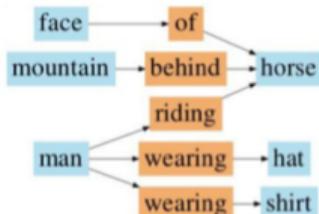
"two young girls are playing with lego toy."

Scene graph prediction



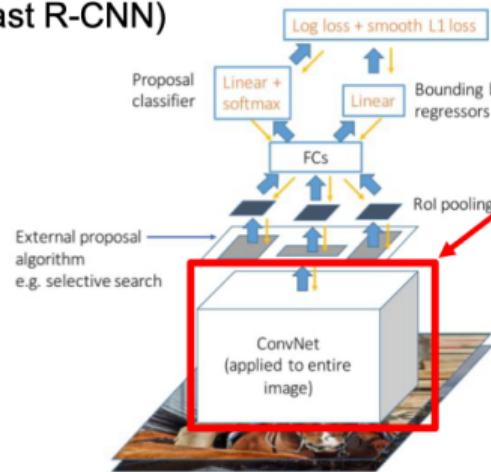
...

Many,  
many  
more



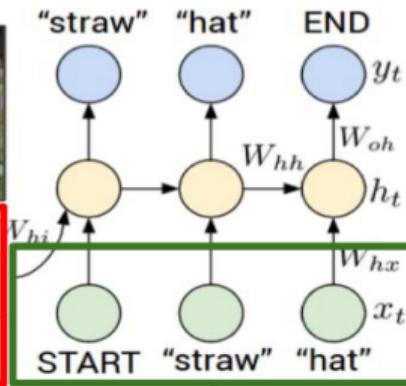
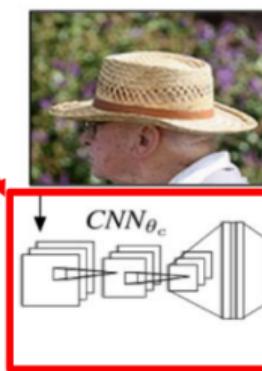
# Transfer Learning With CNNs Is Pervasive...

## Object Detection (Fast R-CNN)



CNN pretrained  
on ImageNet

Image Captioning: CNN + RNN



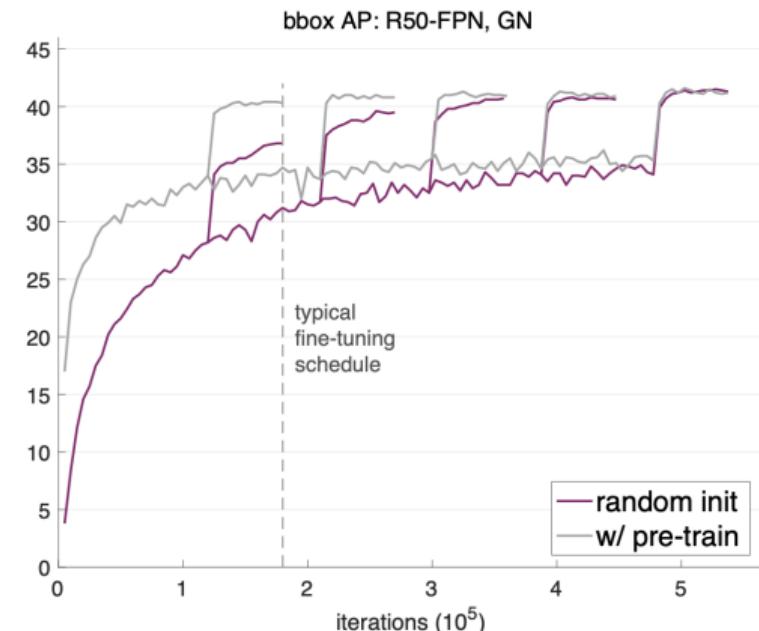
Word vectors pretrained  
with word2vec

Girshick, "Fast R-CNN", ICCV 2015  
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015  
Figure copyright IEEE, 2015. Reproduced for educational purposes.

# Transfer Learning Might Not Always Be Necessary!

- Training from scratch can perform as well as using a pretrained ImageNet model for object detection.
- But it requires 2-3 times longer to train.
- Collecting more data is better than finetuning on a related task



He et al (ICCV 2019)

## Takehome Message

- Have some dataset of interest but it has fewer than ~ 1M images?
    - ① Find a very large dataset that has similar data, train a big ConvNet there
    - ② Transfer learn to your dataset
  - Deep learning frameworks offer a "Model Zoo" of pretrained models so you don't need to train your own
    - **Pytorch:** <https://github.com/pytorch/vision>
    - **TensorFlow:** <https://github.com/tensorflow/models>

## 1 Why Deep Networks?

## 2 AlexNet

## 3 ResNet

## 4 Data Preprocessing

## 5 Data Augmentation

## 6 Transfer Learning

## 7 References

Slides by: Ali Bavafa



# Any Questions?