Machine Learning (CE 40477)

Fall 2024

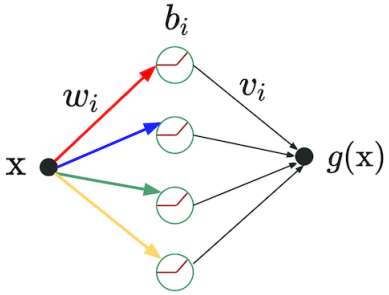
Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

November 19, 2024

Phenomenon Of Deep Expressivity

- While the universal approximation theorem states that a one-hidden-layer neural network can approximate any function to a certain 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** shows that there exist some functions which can be efficiently approximated by a deep network. But to approximate these same functions with a one-hidden layer network would require exponentially many more neurons.



Depth Separation Theorem

- Let any integer $k \geq 1$ and any dimension $d \geq 1$ be given. There exists $f : \mathbb{R}^d \to \mathbb{R}$ computed by a neural network with standard ReLU gates in $2k^3 + 8$ layers, $3k^3 + 12$ total nodes, and $d + 4$ distinct parameters so that:

$$\inf_{g \in \mathscr{G}} \int_{[0,1]^d} |f(x) - g(x)| dx \geq \frac{1}{64}$$

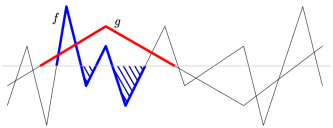
- For any integer $k > 0$, there exist neural networks with $\Theta(k^3)$ layers, $\Theta(1)$ nodes per layer, and $\Theta(1)$ distinct parameters which can not be approximated by networks with $O(k)$ layers unless they are exponentially large (they must possess $\Omega(2^k)$ nodes neurons).

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

Depth Separation Theorem

- This theorem can be proved by showing that:
  1. Functions with few oscillations poorly approximate functions with many oscillations.
  2. Functions computed by networks with few layers must have few oscillations.
  3. Functions computed by networks with many layers can have many oscillations.

- The key idea is that just a few function compositions (layers) suffice to construct a highly oscillatory function, whereas adding nodes without increasing depth results in a function with fewer oscillations. Thereafter, an elementary counting argument suffices to show that low-oscillation functions can't approximate high-oscillation functions.

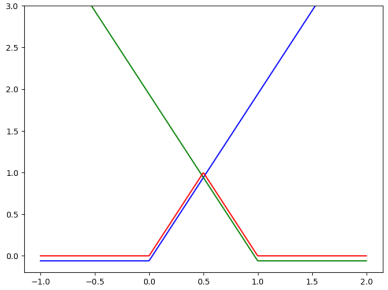


f crosses more than g

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

Intuition

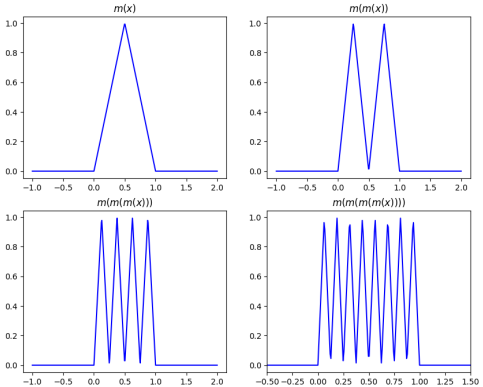
- Let's consider ReLU networks. Let $m(x)$ be the piecewise linear function that is $2x$ if $x \in [0, 0.5]$, $2 - 2x$ if $x \in [0.5, 1]$, and 0 otherwise. This can be computed exactly by a two- or three-layer network (depending on your notational preferences), by the expression: $\sigma(2\sigma(x) - 4\sigma(x - 0.5))$.
- Or in terms of the activation function:



Function m is slightly offset for visual clarity

Intuition

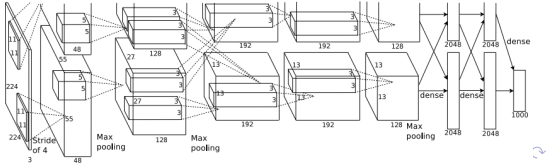
- What do iterates of $m(x)$ look like?
- So $m^{(n)}(x)$ would have 2^n teeth
- We can represent this function with $3n+1$ nodes in a deep network. However, it takes many nodes to do this in a shallow network. Intuitively, depth increases the number of oscillations multiplicatively (which is why we have an exponential number of teeth) whereas width can only do so additively.



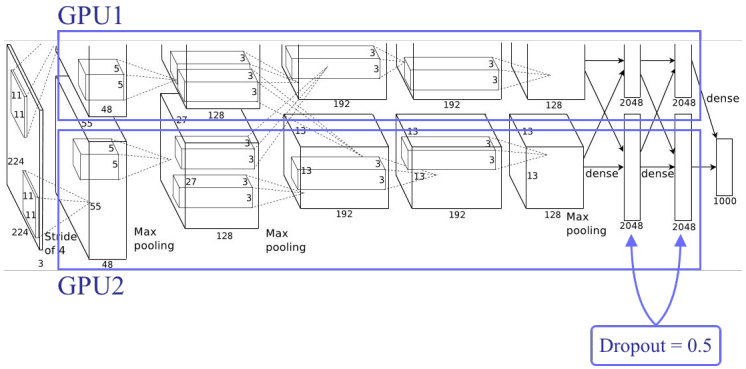
AlexNet [Krizhevsky, et al. NIPS (2012)]

Architecture:

[227 × 227 × 3] INPUT
[55 × 55 × 96] CONV1: 96 11 × 11 filters at stride 4, pad 0
[27 × 27 × 96] MAX POOL1: MAX POOL1: 3 × 3 filters at stride 2
[27 × 27 × 96] NORM1: Normalization layer
[27 × 27 × 256] CONV2: 256 5 × 5 filters at stride 1, pad 2
[13 × 13 × 256] MAX POOL2: 3 × 3 filters at stride 2
[13 × 13 × 256] NORM2: Normalization layer
[13 × 13 × 384] CONV3: 384 3 × 3 filters at stride 1, pad 1
[13 × 13 × 384] CONV4: 384 3 × 3 filters at stride 1, pad 1
[13 × 13 × 256] CONV5: 256 3 × 3 filters at stride 1, pad 1
[6 × 6 × 256] MAX POOL3: 3 × 3 filters at stride 2
[4096] FC6 4096 neurons
[4096] FC7 4096 neurons
[1000] FC8 1000 neurons (class scores)



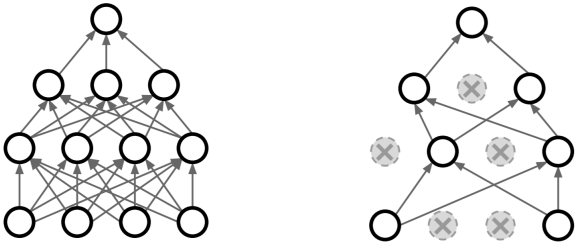
AlexNet At A Glance



- 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 set some neurons to zero.
  - → Probability of dropping is a hyperparameter ⇒ 0.5 is common

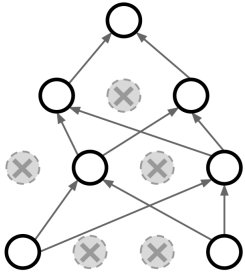


Srivastava et al., JMLR 2014

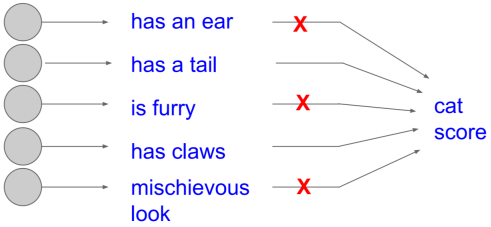
- Thus, a smaller network is trained for each sample.
  - → Smaller network provides a regularization effect.

Dropout

- Why is dropout effective?



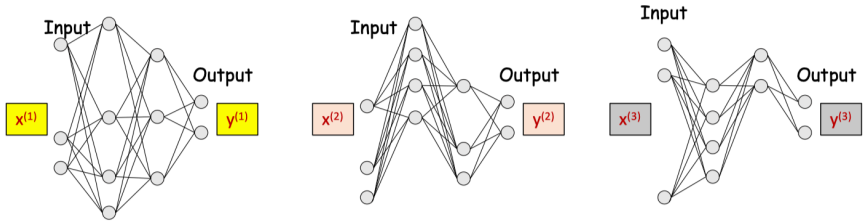
Forces the network to have a redundant representation;
Prevents co-adaptation of features



Hinton et al., 2012

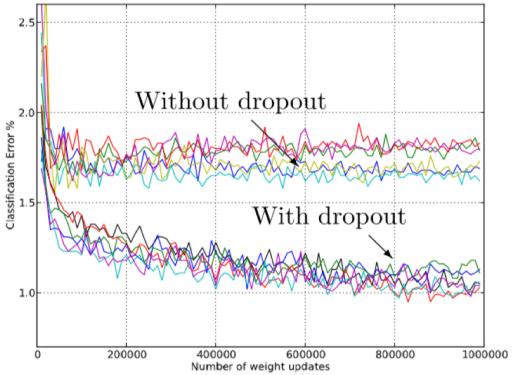
Dropout

- **Training**: Backpropagation is effectively performed only over the remaining network
  - → The resulting effective network varies depending on the input.
  - → Gradients are computed only for the weights and biases between active nodes.
  - → For the remaining, the gradient is just 0.



Dropout

- Test error of different architectures on MNIST with and without dropout
  - → 2-4 hidden layers with 1024-2048 units

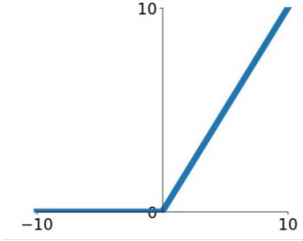


Srivastava et al., 2013

ReLU

ReLU

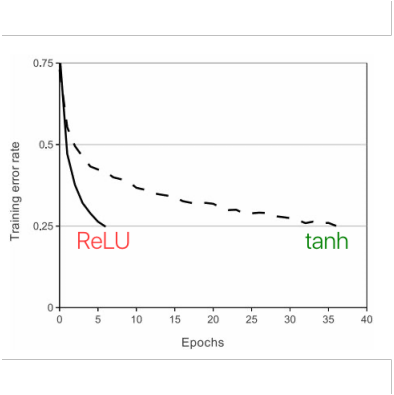
- Does not saturate (in the positive region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid
- **Problems:**
  - → Not zero-centered output
  - → An annoyance:
    - ⋆ Hint: "what is the gradient when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



ReLU

ReLU

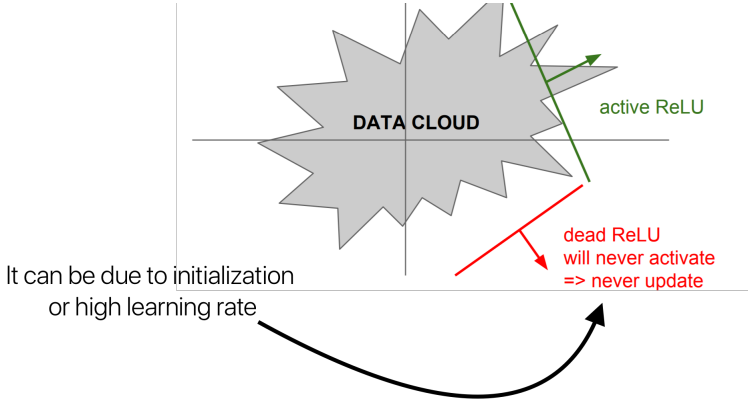
- ReLu trains faster than tanh.



Krizhevsky, et al., 2012

ReLU Initialization

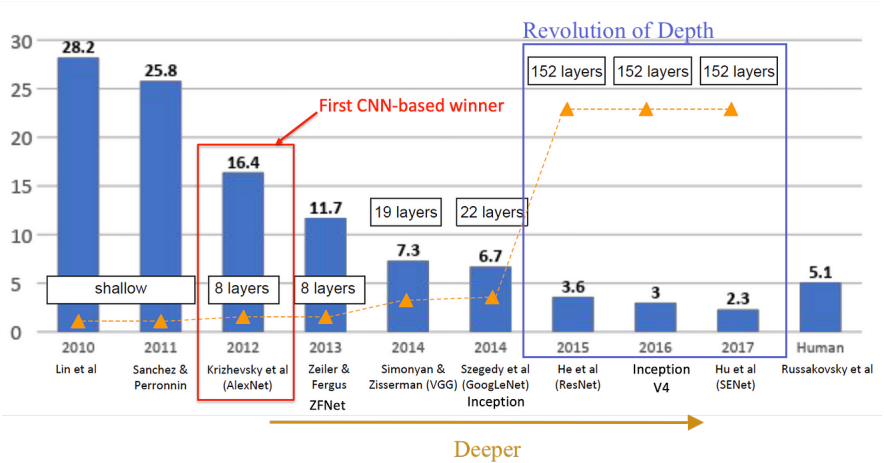
- It is common to initialize ReLU neurons with slightly positive biases (e.g. 0.01).
- It can be due initialization or high learning rate.



Learning Magic In Alexnet

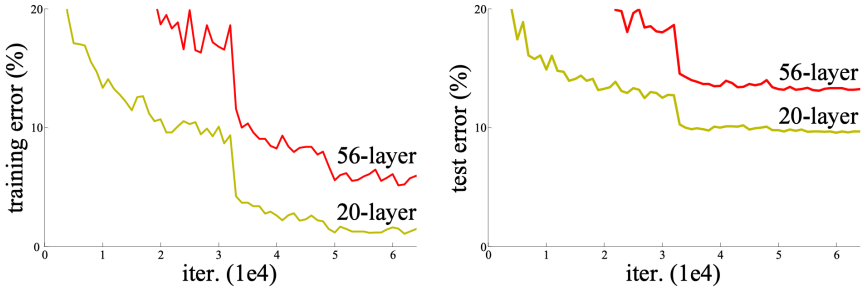
- First use of ReLU
  - → Made a large difference in convergence
- "Dropout" 0.5 (in FC layers only)
- Heavy data augmentation
- SGD with momentum of 0.9 and a mini-batch size of 128
- L2 weight decay 5e-4
- Learning rate: 0.01, decreased by 10 every time validation accuracy plateaus
- Evaluated using: 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 %**

Revolution Of Depth



Challenge Of Making CNNs Deeper

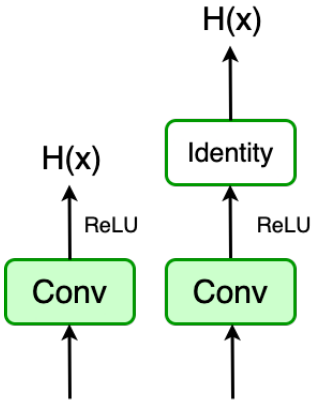
- What happens when we continue stacking deeper layers on a “plain” convolutional neural network?
- **Question**: Whats strange about these training and test curves?



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

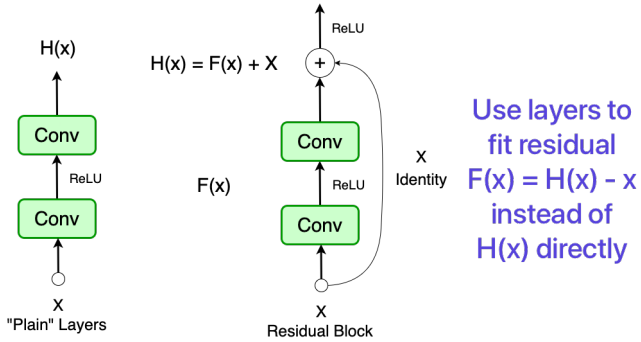
Challenge Of Making CNNs Deeper

- **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 its not caused by overfitting!
- **Hypothesis:** The issue is related to optimization, deeper models are harder to optimize.
- What must the deeper model learn 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.



Residual Block

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



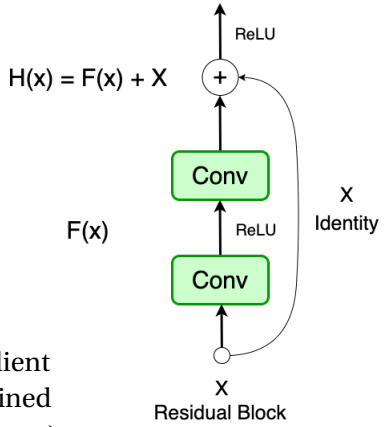
Backprop In Residual Block

- How exactly does the skip connection prevent gradient vanishing?
- **During backpropagation, there are two pathways** for the gradient to flow back to the input layer while traversing a residual block

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

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

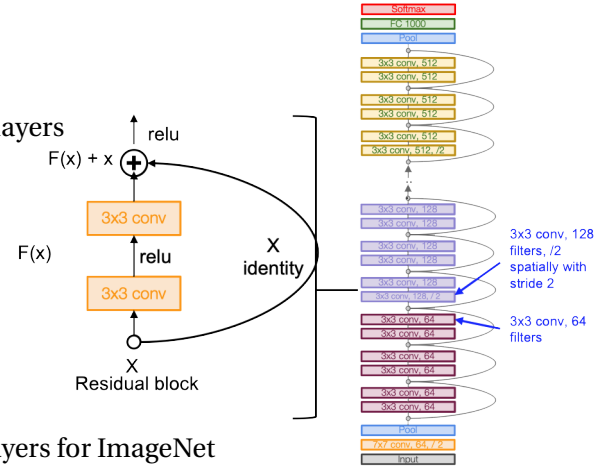
- As we can see from the above equation, even if the gradient of Path 1 vanished to 0, the gradient of Path 2 still remained (because this gradient does not encounter any weight layer)



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

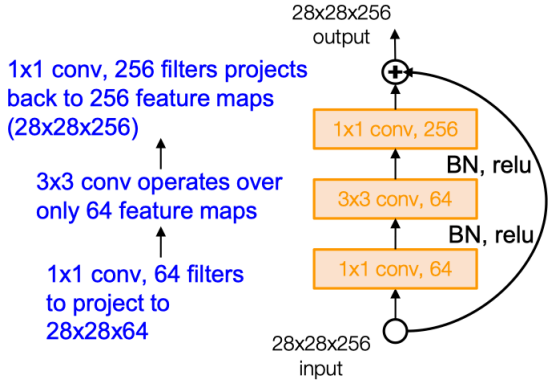
Architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double number of filters and downsample spatially using stride 2 (/2 in each dimension) Reduce the activation volume by half.
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)
- Total depths of 18, 34, 50, 101, or 152 layers for ImageNet



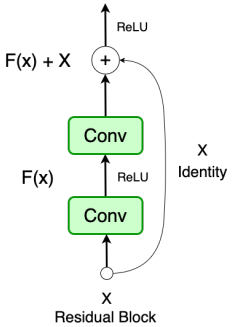
Deeper Residual Module (Bottleneck)

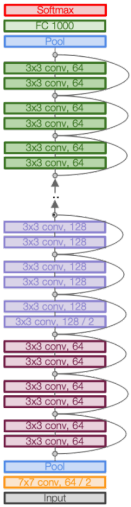
- For deeper networks (ResNet- 50+), use bottleneck layer to improve efficiency.



ResNet At A Glance

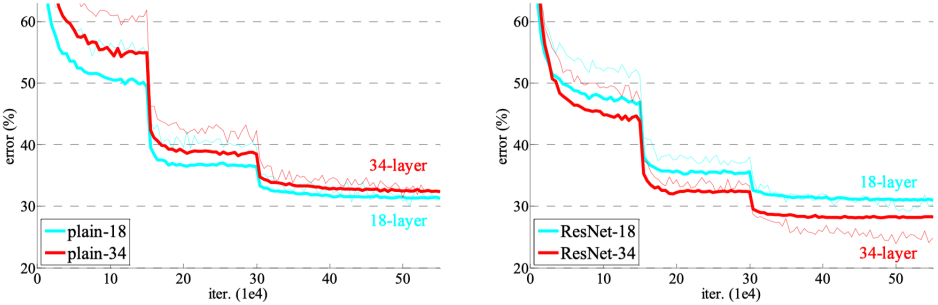
- Very deep networks using residual connections
- ResNet **introduces** the concept of skip connections (or residual connections) that allow the gradient to be directly backpropagated to earlier layers.
- **Skip connections helps overcoming gradient vanishing** problem, where the accuracy saturates and then degrades rapidly as the network depth increases.
- 152-layer model for ImageNet
- ILSVRC15 classification winner (3.57 % top 5 error)
- Swept all classification and detection competitions in ILSVRC15 and COCO15!





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:

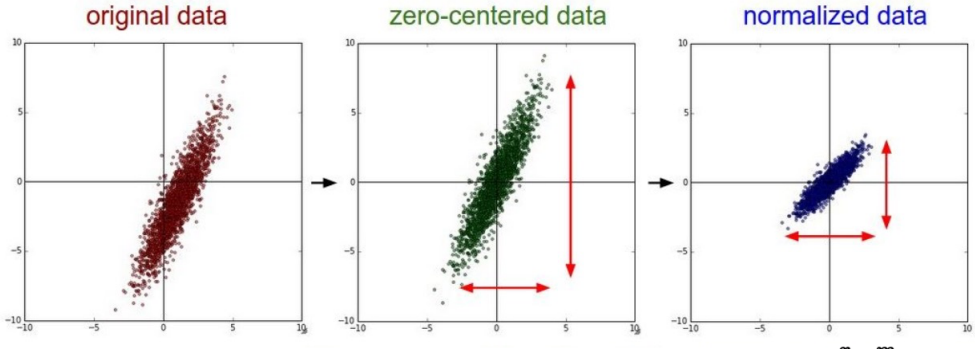
- Batch Normalization after every CONV layer
- Xavier initialization from He, et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

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
- 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

Normalizing The Data

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

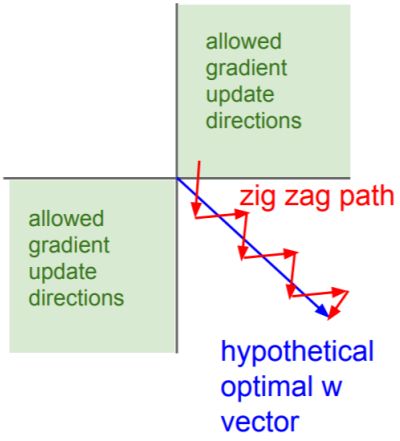


TLDR: In Practice For Images: Center Only

- 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)

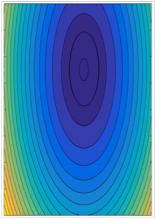
Why Zero-Mean The Input?

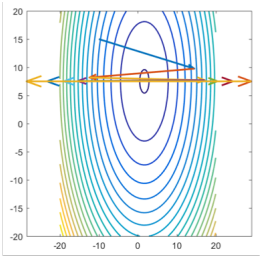
- Reminder: sigmoid
- Consider what happens when the input to a neuron is always positive...
- What can we say about the gradients on w? Always all positive or all negative
  - This is also why zero-mean data is preferable!



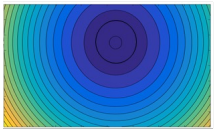
Why Normalize The Input?

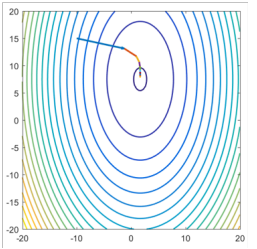
- Very different range of input features
- i.e. poor conditioning
- Causes noisy movement of gradient



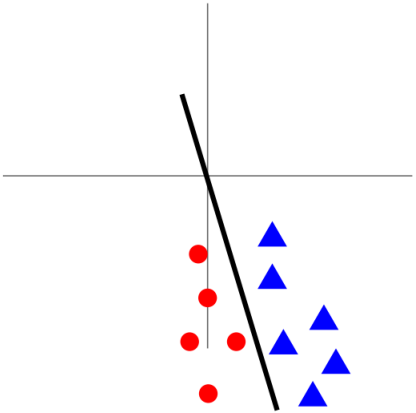


- Normalized data:
  - Improves loss landscape (alleviate poor conditioning)
  - Faster optimization (training)

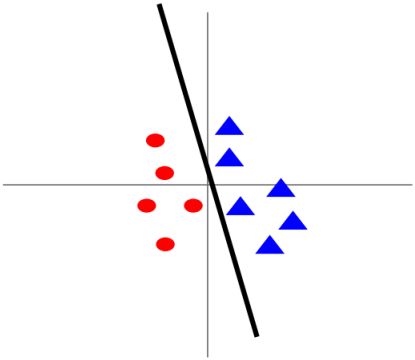




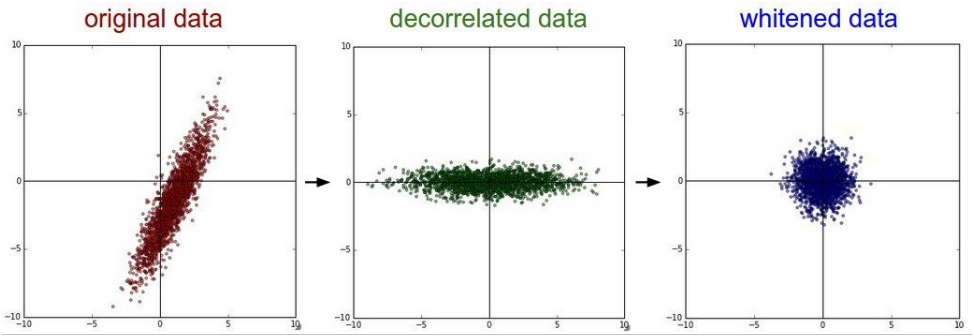
Before Normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After Normalization: loss is less sensitive to small changes in weight matrix; easier to optimize

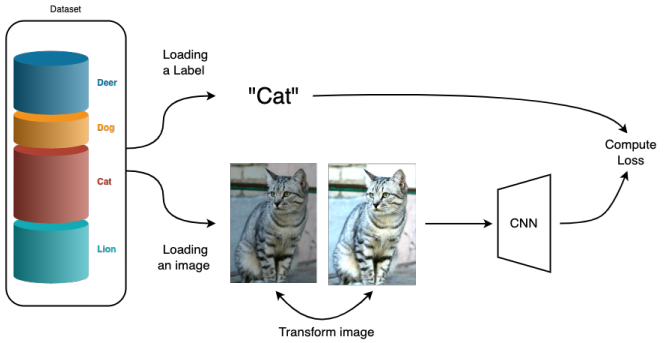


Not Common To Normalize Variance, To Do PCA Or Whitening



Motivation

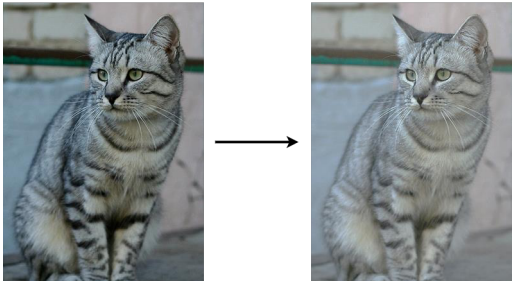
- Getting more training data can be expensive
- But, we can often generate additional training examples from existing datasets.
- Transform each input data example in such a way that the label stays the same
- Benefits:
  - Reduces overfitting
  - Improves generalization
  - Acts as a regularization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



Augmentations In Action

- Simple
  - Randomize contrast and brightness
- 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)

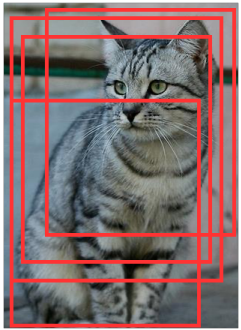
Color Jitter



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
- Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

Cutout

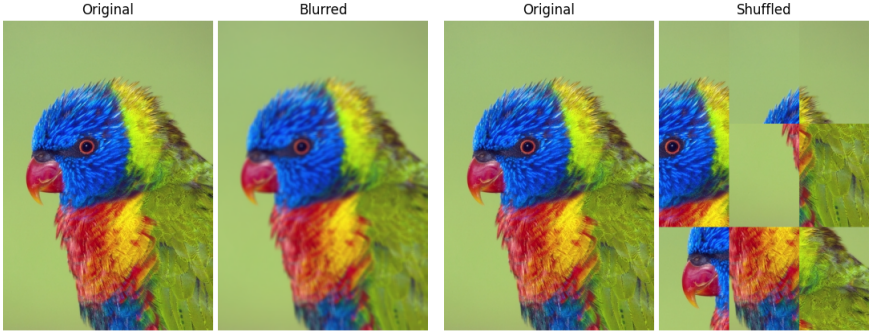
DeVries and Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout", arXiv 2017

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



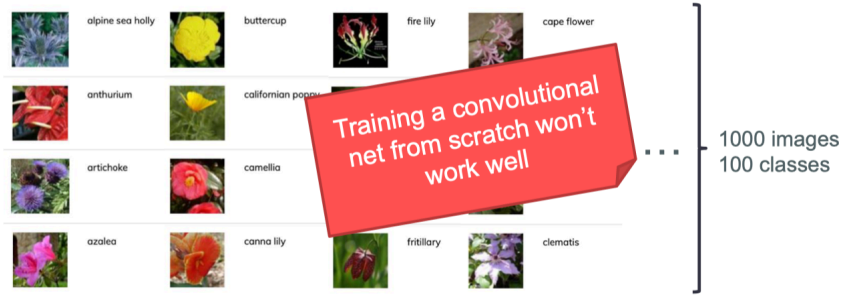
`A.GaussianBlur()`
Blurs using Gaussian kernel

`A.RandomGridShuffle()`
Divides into a grid and shuffles the blocks

Image source: freepik.com

Motivation

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



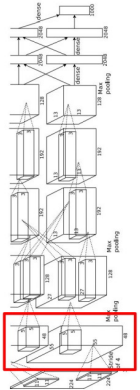
https://www.robots.ox.ac.uk/ vgg/data/flowers/102

Transfer learning

- However, by transfer learning you can use the learned features for a task (using a large amount of data) in another related task (for which you don't have enough data).
- **How should we train?**
- Option 1: Train only classification layer, freeze backbone.
  - → Often referred to as the "linear evaluation protocol"
  - → Fast & simple
- Option 2: Train classification layer, fine-tune backbone at the same time.
  - → Slower, but can adapt feature extraction to dataset statistics

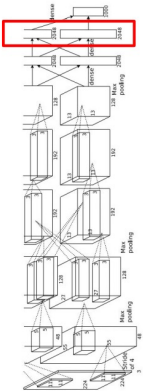
Transfer Learning: The idea

Earlier Layers -> more local features

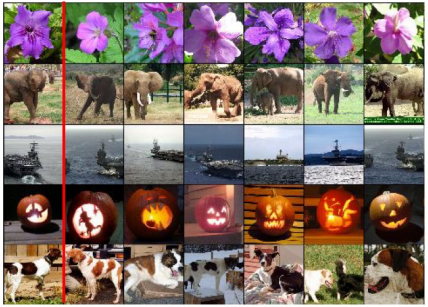


AlexNet:
64 x 3 x 11 x 11

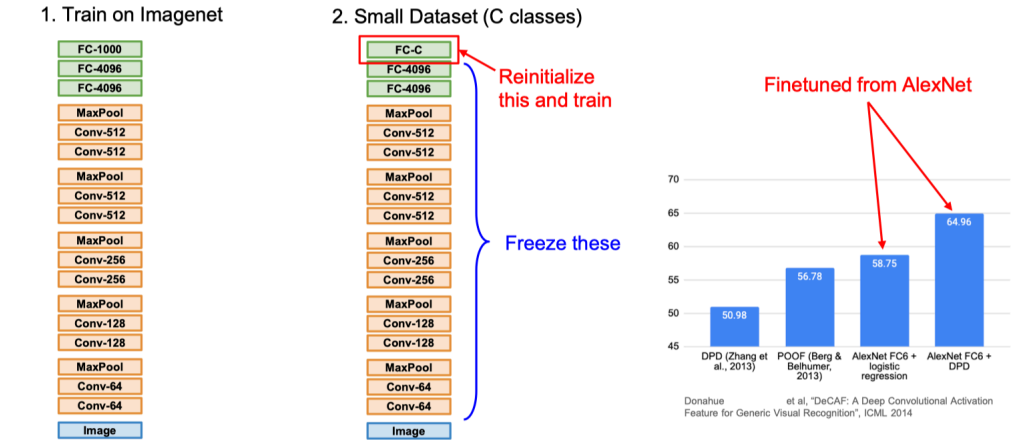
Final Layers -> more global features



Test image L2 Nearest neighbors in feature space

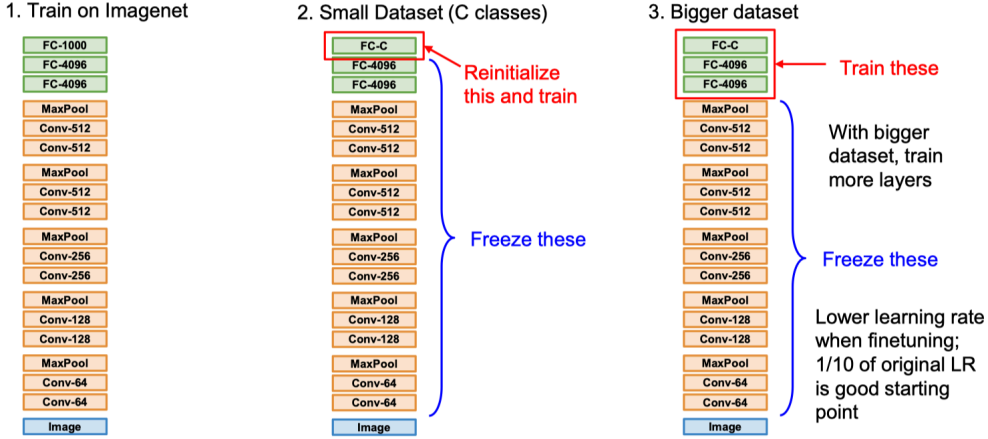


Transfer Learning With CNNs

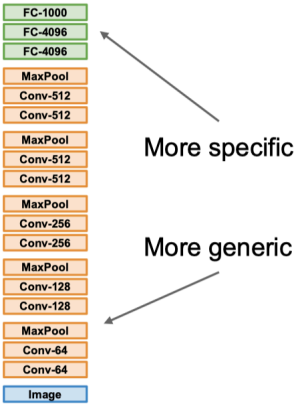


Donahue et al, DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition, ICML 2014
Razavian et al, CNN Features Off-the-Shelf: An Astounding Baseline for Recognition, CVPR Workshops 2014

Transfer Learning With CNNs

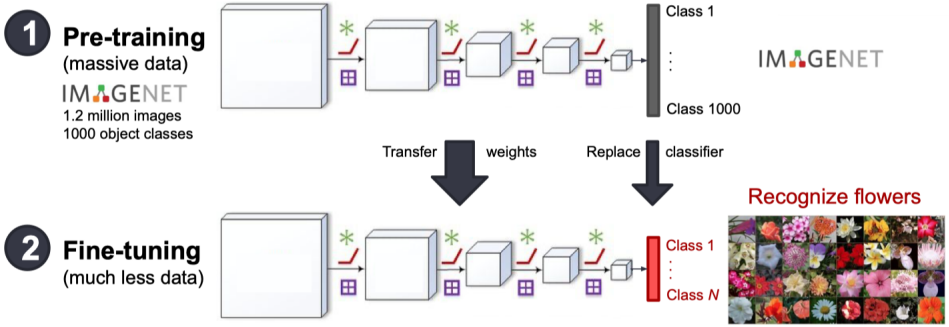


Transfer Learning With CNNs

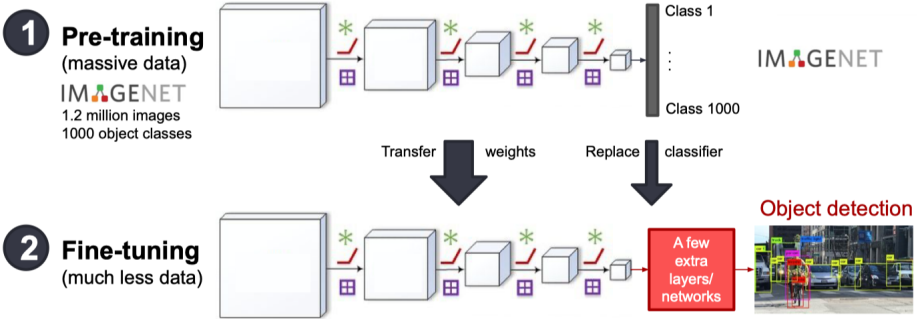


	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers or start from scratch!

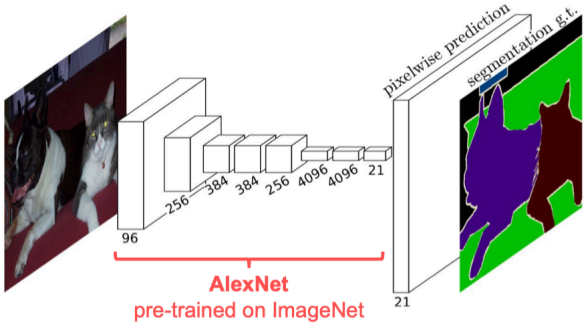
Transfer Learning With CNNs



Transfer Learning: Domain Shift



Example: Semantic Segmentation



Long, Shelhamer, Darrell, CVPR 2015

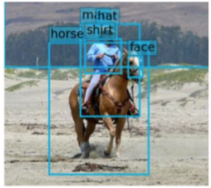
Transfer Learning With CNNs Is Pervasive

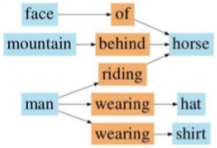
Pose estimation

Image captioning

"two young girls are playing with lego toy."

Scene graph prediction

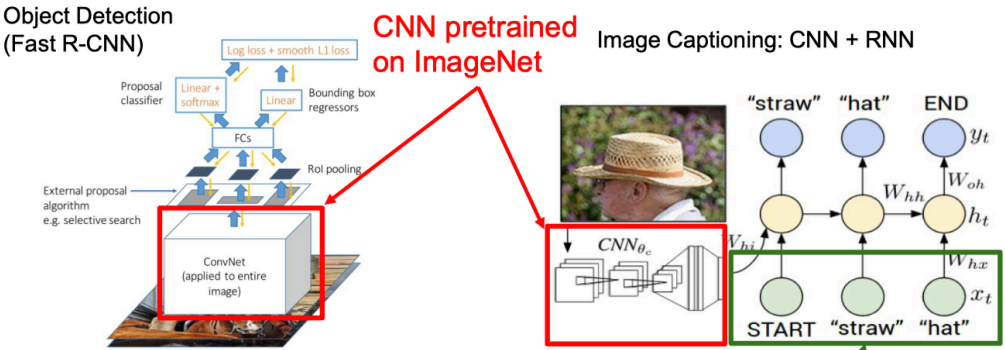




...

Many, many more

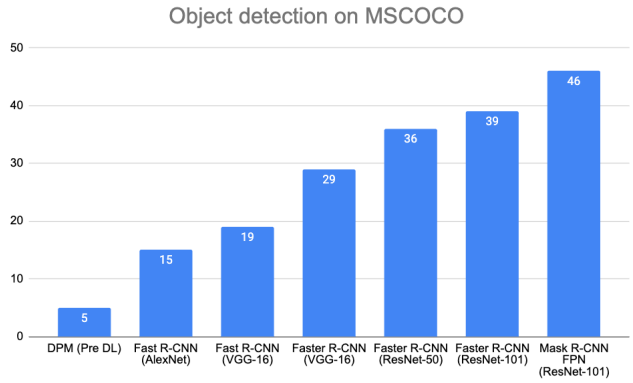
Transfer Learning With CNNs Is Pervasive



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.

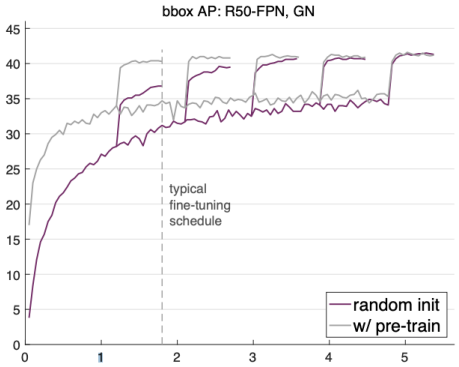
Architecture Matters In TL!



Girshick, The Generalized R-CNN Framework for Object Detection, ICCV 2017

Transfer Learning Might Not Always Be Necessary!

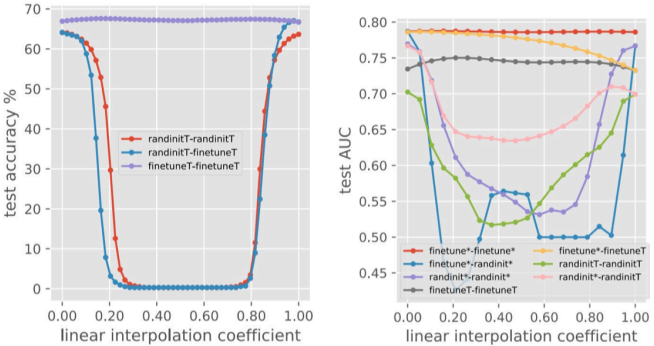
- Training from scratch can perform as well as using a pretrained ImageNet model for object detection.
- But it takes 2-3x as long to train.
- Collecting more data is better than finetuning on a related task



He et al, Rethinking ImageNet Pre-training, ICCV 2019 Figure copyright Kaiming He, 2019. Reproduced with permission.

Performance Barrier Between Different Solutions

- There are no performance barriers between finetuned models.
- Finetuned models reside in the same basin
- Random initializations end up in different basins, even for the same random seed.



He et al, Rethinking ImageNet Pre-training, 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

Slides by: Ali Bavafa

[1] Fei Fei Li, “Stanford University cs231n.” Lecture slides.

[2] M. Soleymani Baghshah, “Machine learning.” Lecture slides.

[3] B. Raj, “Carnegie Mellon University 11-785.” Lecture slides.

[4] Alexander Ecker, “Neuromatch Academy.” Lecture slides.

[5] MIT, “Mathematical aspects of deep learning,” 2017. `https://elmos.scripts.mit.edu/mathofdeeplearning/`.

Any Questions?