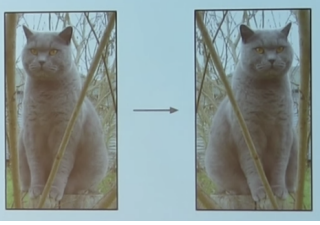
practical advice for working with images

Data Augmentation

1. Horizontal flips



Data Augmentation

2. Random crops/scales

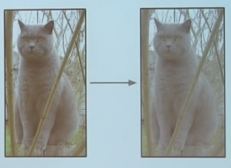
Training: sample random crops / scales
ResNet:
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

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

Data Augmentation

3. Color jitter

Simple:
Randomly jitter contrast



Complex:

1. Apply PCA to all [R pixels in training se
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012]*, ResNet, etc)

Data Augmentation

4. Get creative!

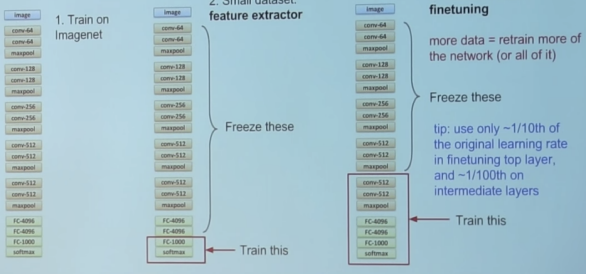
Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

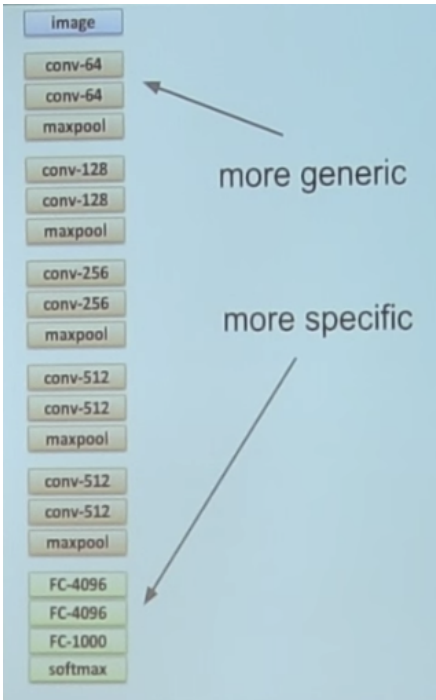
Transfer Learning

"You need a lot of data if you want to train/use CNNs"

BUSTED

Transfer Learning with CNNs



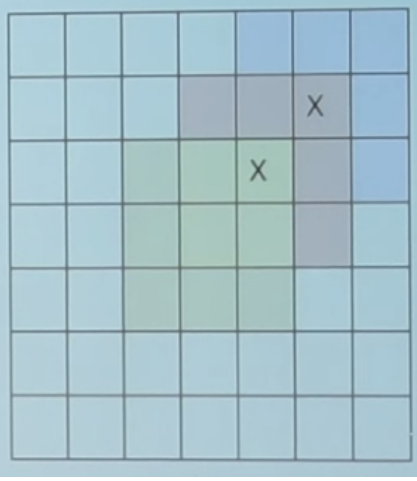


	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7



Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters	three CONV with 3 x 3 filters
Number of weights: $= C \times (7 \times 7 \times C) = 49\ C^2$	Number of weights: $= 3 \times C \times (3 \times 3 \times C) = 27\ C^2$
Number of multiply-adds: $= (H \times W \times C) \times (7 \times 7 \times C)$ $= \mathbf{49\ HWC^2}$	Number of multiply-adds: $= 3 \times (H \times W \times C) \times (3 \times 3 \times C)$ $= \mathbf{27\ HWC^2}$

- Also, if we stick ReLUs in between these more layers but smaller convs, we get more non-linearity in our model for the same computations.
- This is the theme of tricks used in inception modules. Save on number of params and computations while getting more expressive power!
- We can take convolutional layers and express them as matrix multiplies. this is really good from the perspective of taking advantage of matrix multiply optimizations that have been built for various computing platforms, GPUs being the main ones.

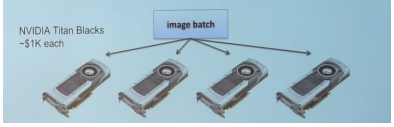
GPUs can be programmed

- CUDA (NVIDIA only)
  - Write C code that runs directly on the GPU
  - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on anything
  - Usually slower :(

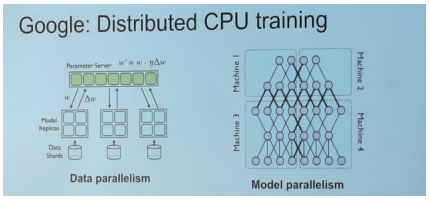
Even with GPUs, training can be slow

VGG: ~2-3 weeks training with 4 GPUs

ResNet 101: 2-3 weeks with 4 GPUs



Google: Distributed CPU training



GPU - CPU communication is a bottleneck.

=>

CPU data prefetch+augment thread running

while

GPU performs forward/backward pass

Floating point precision

- 64 bit "double" precision is default in a lot of programming
- 32 bit "single" precision is typically used for CNNs for performance

Implementation details: Recap

- GPUs much faster than CPUs
- Distributed training is sometimes used
  - Not needed for small problems
- Be aware of bottlenecks: CPU / GPU, CPU / disk
- Low precison makes things faster and still works
  - 32 bit is standard now, 16 bit soon
  - In the future: binary nets?