Lecture 11

- ResNets
- Normalization
- U-nets

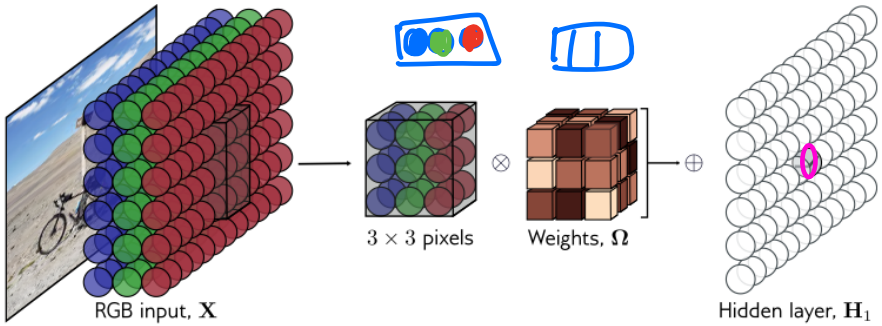


Figure 10.10 2D convolution applied to an image. The image is treated as a 2D input with three channels corresponding to the red, green, and blue components. With a 3×3 kernel, each pre-activation in the first hidden layer is computed by pointwise multiplying the 3×3×3 kernel weights with the 3×3 RGB image patch centered at the same position, summing, and adding the bias. To calculate all the pre-activations in the hidden layer, we "slide" the kernel over the image in both horizontal and vertical directions. The output is a 2D layer of hidden units. To create multiple output channels, we would repeat this process with multiple kernels, resulting in a 3D tensor of hidden units at hidden layer $\mathbf{H}_1$.

So what does a 1x1 conv do?

Common input: 2D image with 3 channels RGB.

To generate 1 hidden layer with a 3x3 kernel

we have $3 \times 3 \times 3$ weight

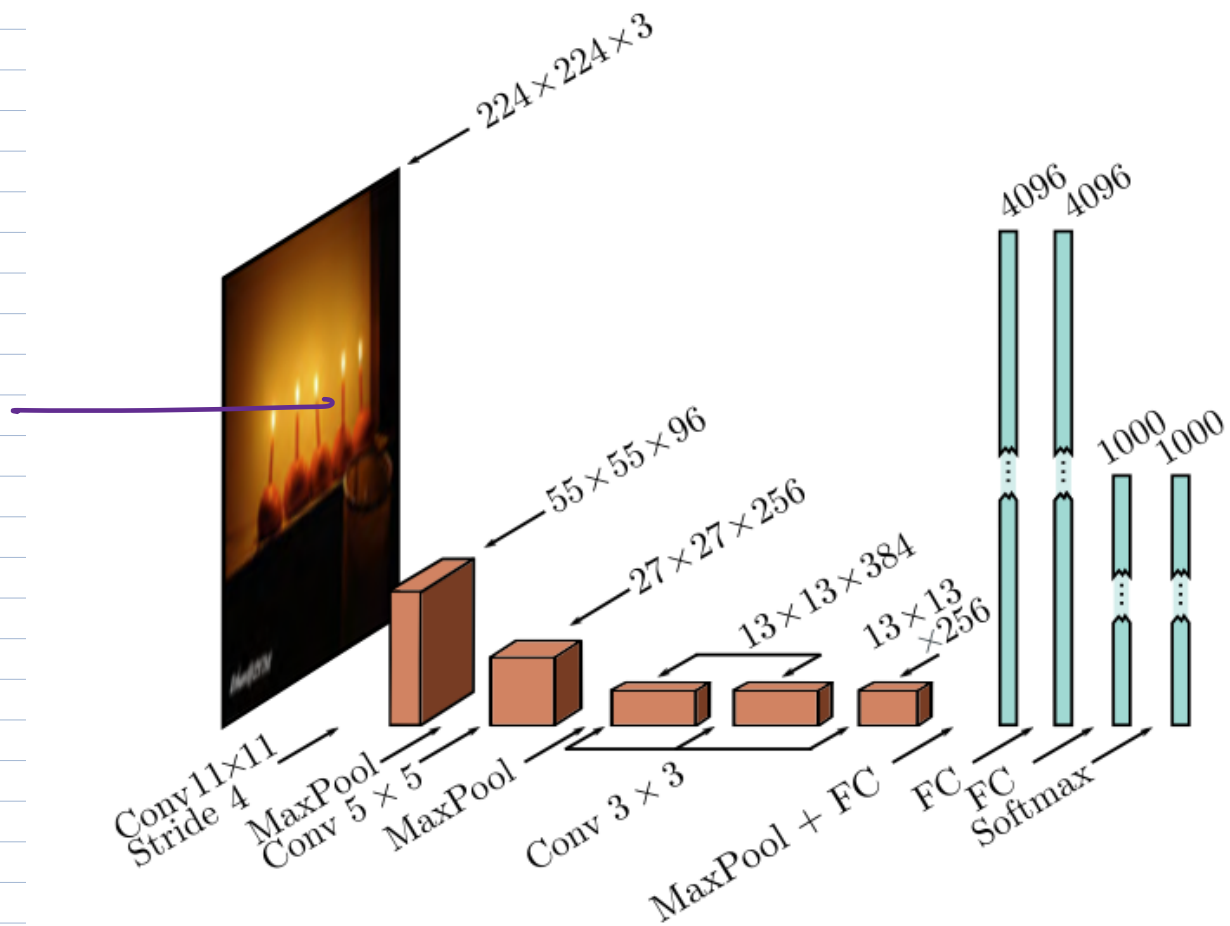
(ch, ht, width)

To create C_o output channels, repeat this C_o times

With C_{in} input channels, $K \times K$ kernel need: $C_{in} \times C_o \times K \times K$ weights.

and C_o biases.

Alex Net ~ 60 million parameters



- Data Augmentation.
- Dropout.
- Weight decay

Dropout

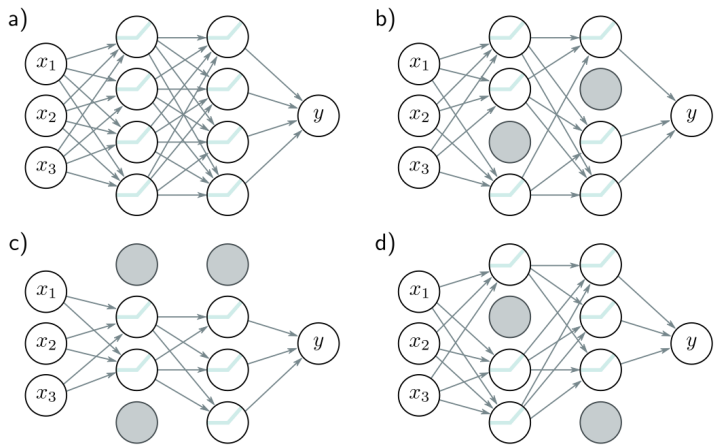


Figure 9.8 Dropout. a) Original network. b–d) At each training iteration, a random subset of hidden units is clamped to zero (gray nodes). The result is that the incoming and outgoing weights from these units have no effect, so we are training with a slightly different network each time.

Data augmentation applied inside our network during training

Idea: Zero out random activations inside network during training

Impact: Encourages redundancy.

$\vec{h}_l$: hidden layer

$$\tilde{h}_l[i] = \begin{cases} 0 & \text{with prob. } 1-p \\ h_l[i] & \text{with prob } p \end{cases}$$

} Do a coin toss ie multiply by Bernoulli(p)

Test time? Scale activations by p

Pytorch: Scale by $\frac{1}{p}$ at train time.

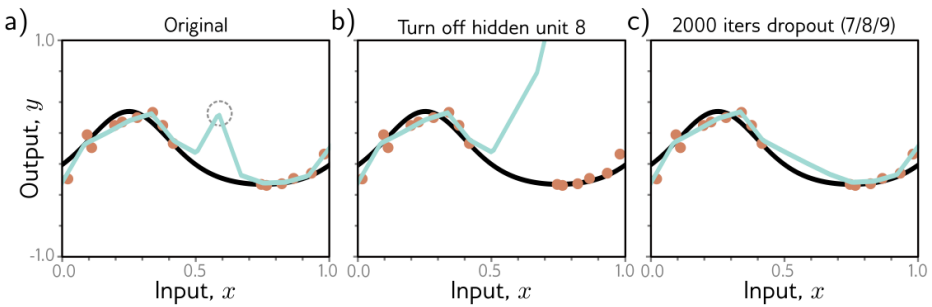
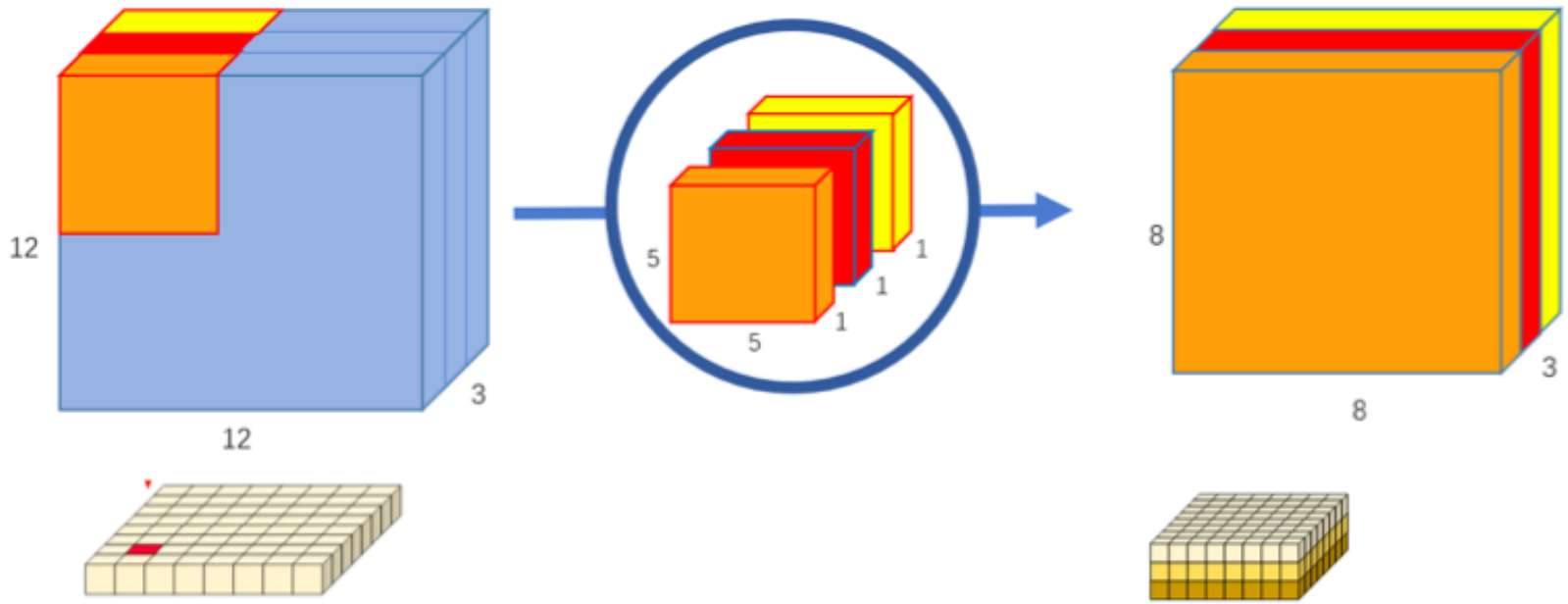


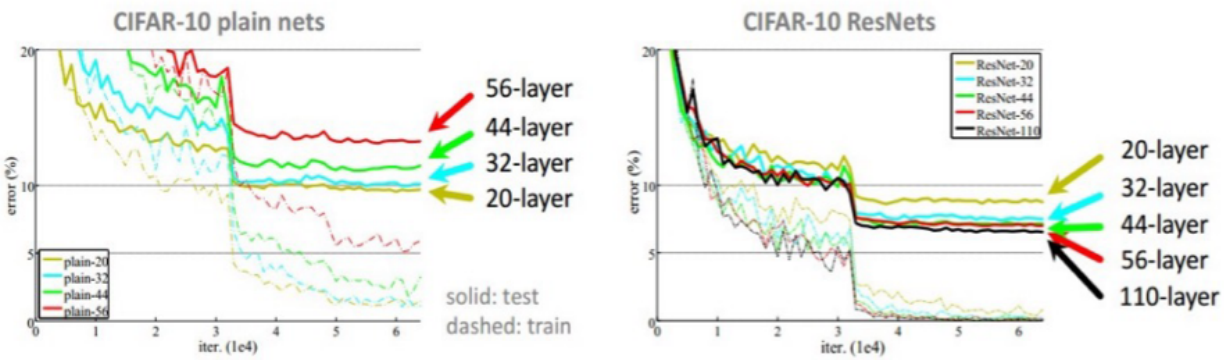
Figure 9.9 Dropout mechanism. a) An undesirable kink in the curve is caused by a sequential increase in the slope, decrease in the slope (at circled joint), and then another increase to return the curve to its original trajectory. Here we are using full-batch gradient descent, and the model (from figure 8.4) fits the data as well as possible, so further training won't remove the kink. b) Consider what happens if we remove the eighth hidden unit that produced the circled joint in panel (a), as might happen using dropout. Without the decrease in the slope, the right-hand side of the function takes an upwards trajectory, and a subsequent gradient descent step will aim to compensate for this change. c) Curve after 2000 iterations of (i) randomly removing one of the three hidden units that cause the kink and (ii) performing a gradient descent step. The kink does not affect the loss but is nonetheless removed by this approximation of the dropout mechanism.

Depthwise convolution



Filters and the input are broken into different channels, convolved separately, then concatenated

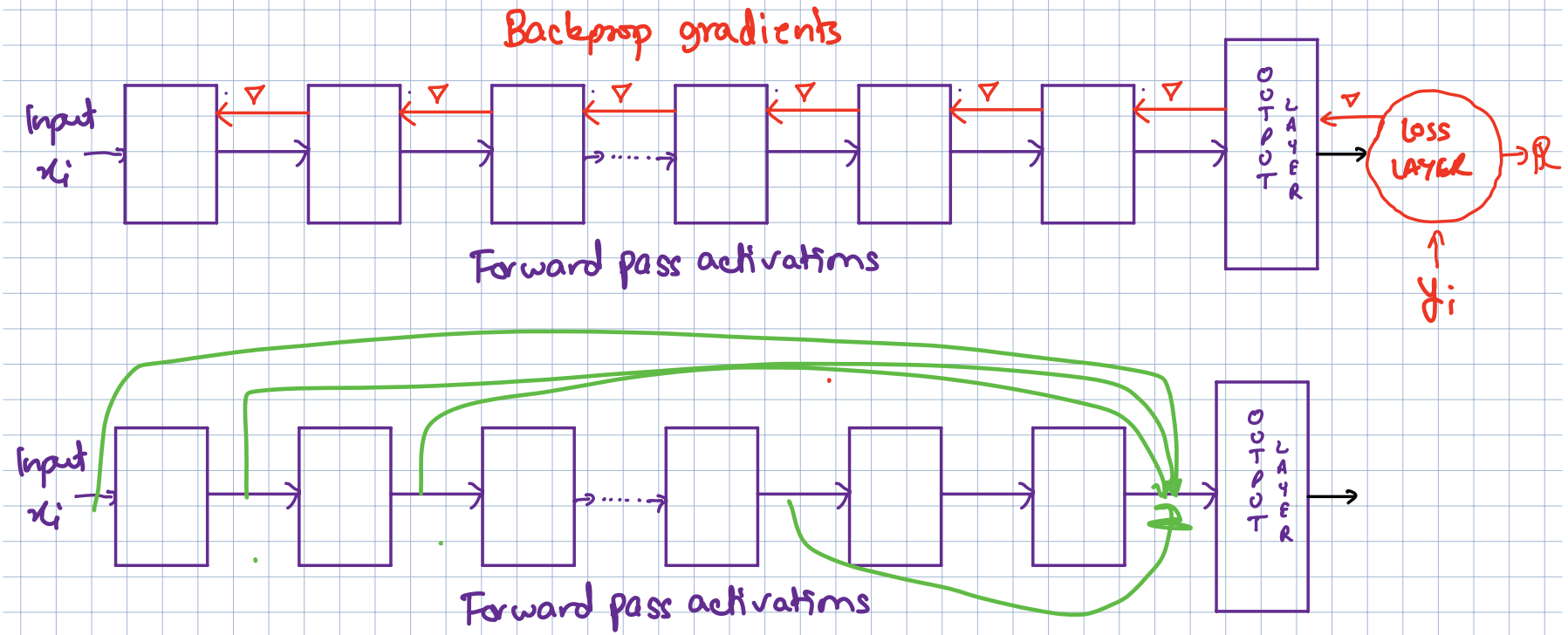
CIFAR-10 experiments



from He et al., and others

Observation: Gradients are going to zero as backprop has to go through more layers.

- Why might depth hurt optimization?



- Dying gradients
- Confused gradients
- Gradient telephone.

Shattered gradients.

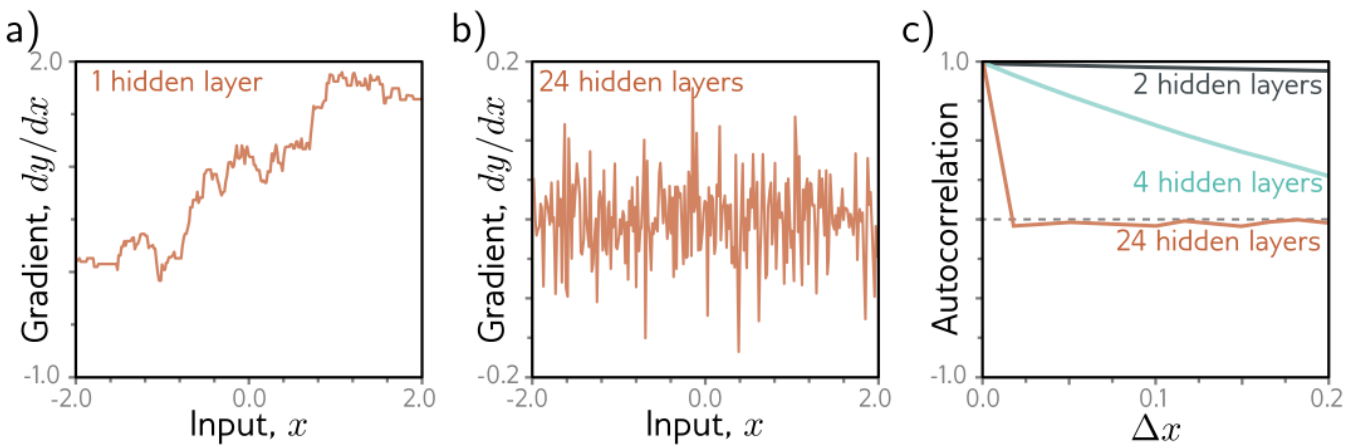
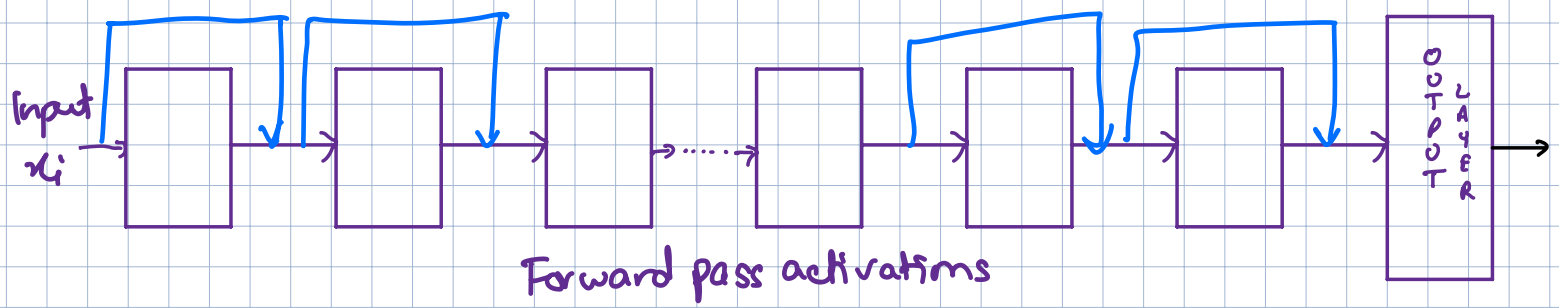


Figure 11.3 Shattered gradients. a) Consider a shallow network with 200 hidden units and Glorot initialization (He initialization without the factor of two) for both the weights and biases. The gradient $\partial y/\partial x$ of the scalar network output y with respect to the scalar input x changes relatively slowly as we change the input x. b) For a deep network with 24 layers and 200 hidden units per layer, this gradient changes very quickly and unpredictably. c) The autocorrelation function of the gradient shows that nearby gradients become unrelated (have autocorrelation close to zero) for deep networks. This *shattered gradients* phenomenon may explain why it is hard to train deep networks. Gradient descent algorithms rely on the loss surface being relatively smooth, so the gradients should be related before and after each update step. Adapted from Balduzzi et al. (2017).



- Roughly doubles trainable depth.

ResNets

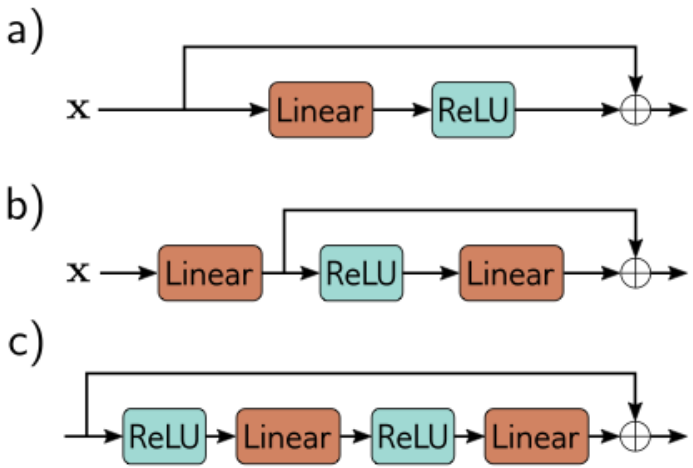


Figure 11.5 Order of operations in residual blocks. a) The usual order of linear transformation or convolution followed by a ReLU nonlinearity means that each residual block can only add non-negative quantities. b) With the reverse order, both positive and negative quantities can be added. However, we must add a linear transformation at the start of the network in case the input is all negative. c) In practice, it's common for a residual block to contain several network layers.

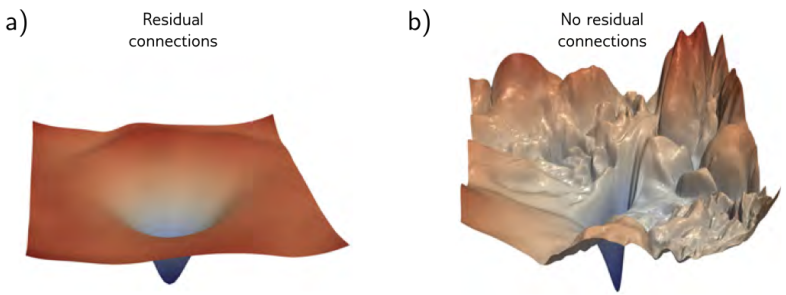
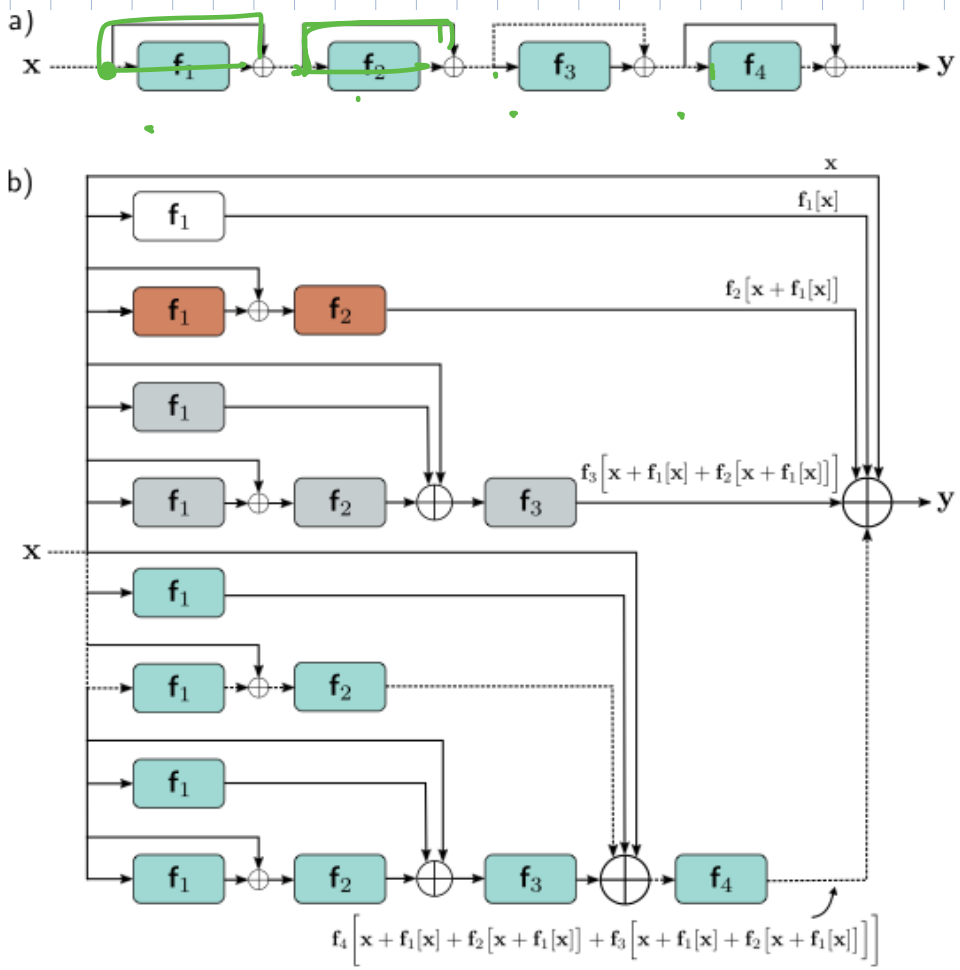


Figure 11.13 Visualizing neural network loss surfaces. Each plot shows the loss surface in two random directions in parameter space around the minimum found by SGD for an image classification task on the CIFAR-10 dataset. These directions are normalized to facilitate side-by-side comparison. a) Residual net with 56 layers. b) Results from the same network without skip connections. The surface is smoother with the skip connections. This facilitates learning and makes the final network performance more robust to minor errors in the parameters, so it will likely generalize better. Adapted from Li et al. (2018b).



- 16 different paths
  2×2×2×2

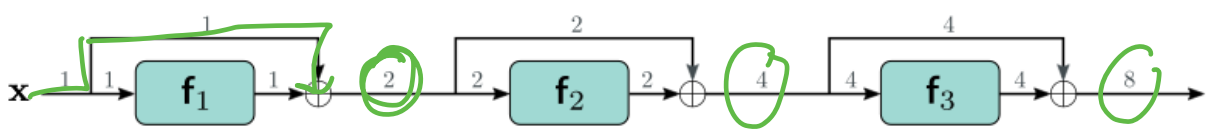
Pre-res net: ith output: $f_i(f_{i-1}(\cdots f_1(x)))$

Res net perspective: $x + g(x) \rightarrow$ ith output

New block:
Δ or change in function
$\frac{d}{dx}(\ ) = f_i(x)$.

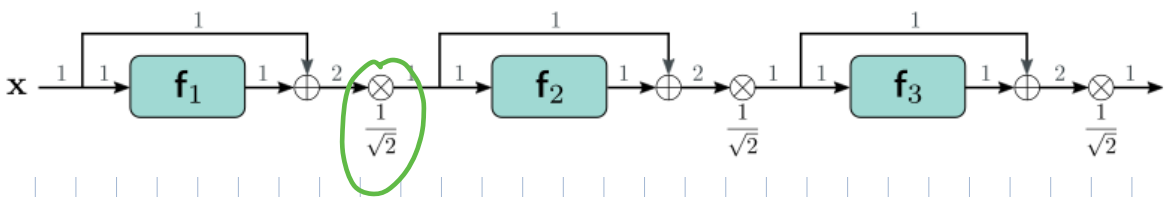
What is the challenge?

a)



Strategy 1:

b)



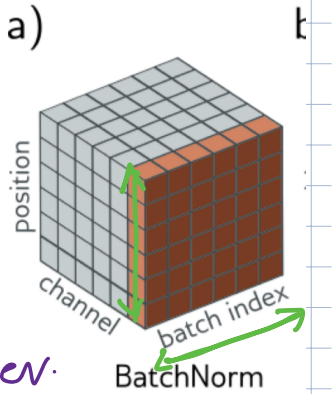
Batch Norm

$$m_h = \frac{1}{|B|} \sum_{i \in B} h_i$$

mean

$$s_h = \sqrt{\frac{1}{|B|} \sum_{i \in B} (h_i - m_h)^2}$$

standard dev.



$$h_i \leftarrow \frac{h_i - m_h}{s_h + \varepsilon}$$

$\varepsilon > 0$ small

$$h_i \leftarrow \gamma \cdot h_i + \delta$$

γ and δ are learned parameters.

K layers, D units per layer — $2KD$ new parameters

Conv networks: Avg. over spatial positions

K layers. C channels per layer

KC offsets + KC scales.

Batch norm pros and cons

- Initialize $\delta = 0$, $\gamma = 1$
  Stable forward propagation
- Can use higher learning rates
- Regularization
  → Normalization depends on batch statistics.
- Decreases variation in loss and gradient

Other normalizations

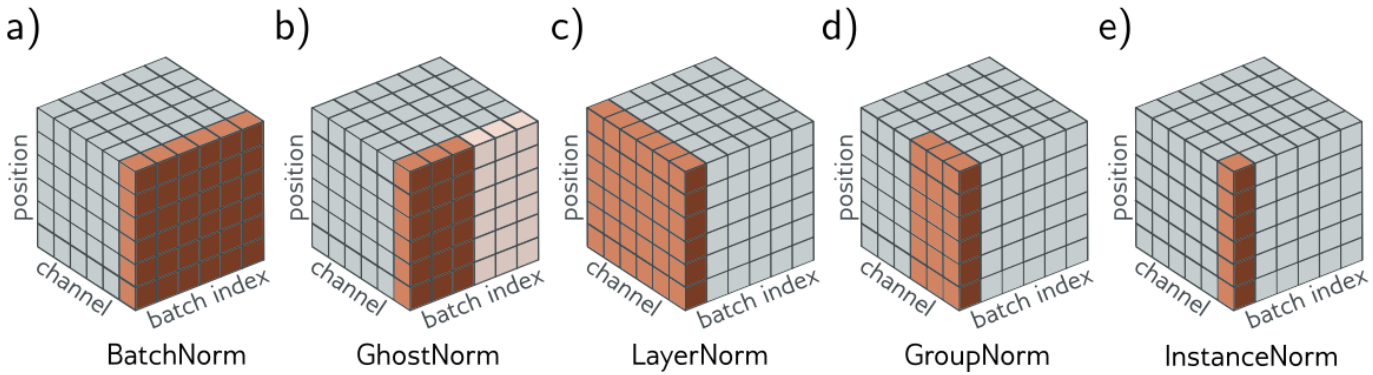


Figure 11.14 Normalization schemes. BatchNorm modifies each channel separately but adjusts each batch member in the same way based on statistics gathered across the batch and spatial position. Ghost batch normalization computes these statistics from only part of the batch to make them more variable. LayerNorm computes statistics for each batch member separately, based on statistics gathered across the channels and spatial position. It retains a separate learned scaling factor for each channel. GroupNorm normalizes within each group of channels and also retains a separate scale and offset parameter for each channel. InstanceNorm normalizes within each channel separately, computing the statistics only across spatial position. Adapted from Wu & He (2018).

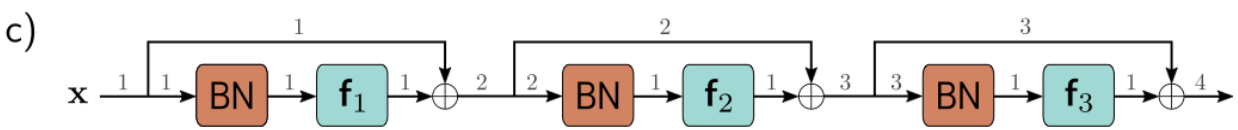
GhostNorm: Batch norm using only subset of batch

Layer Norm: Average over all pixels + channels

Group Norm: Divides channels into groups.

Instance Norm: Separately for each channel

Where do you put norm block?



Pre-norm convention. Inside each block at the beginning.

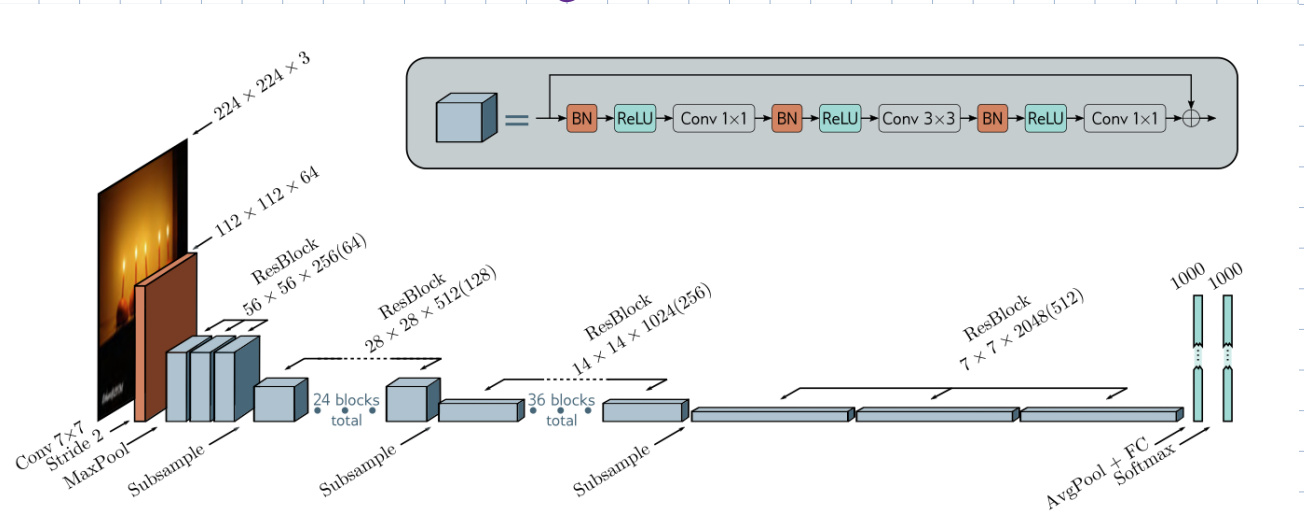
Normalization a test time?

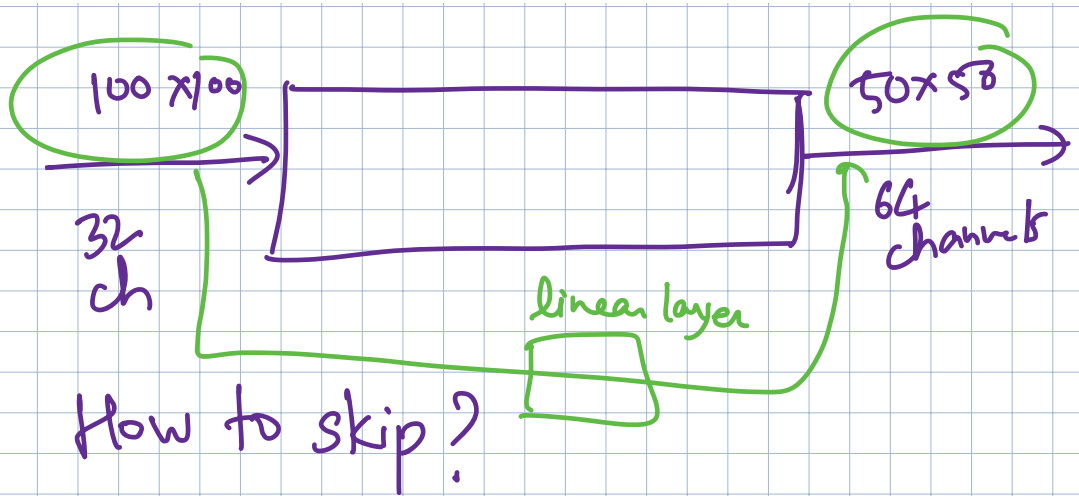
Layer Norm: No problem

Batch Norm: Avg over all data to compute m, s.
Held out data.

- Aug across all data
- Use held out data to learn m, s
-

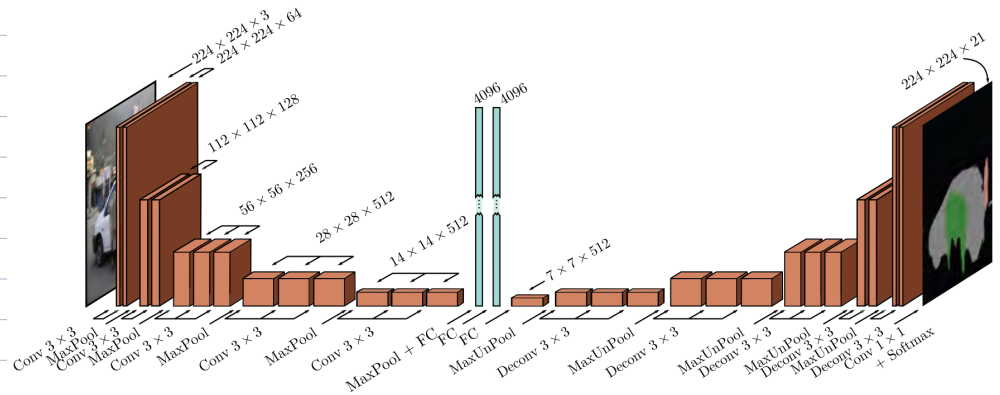
Another challenge





How to do image-level tasks?



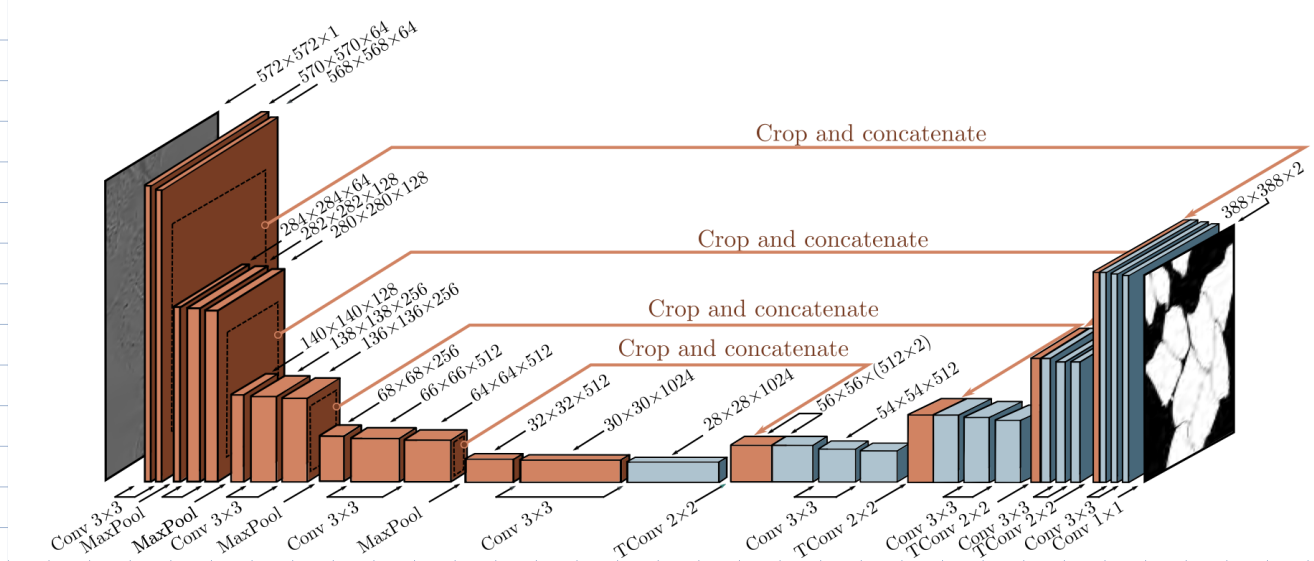


Transpose Convolution

Max-unpooling / upsampling

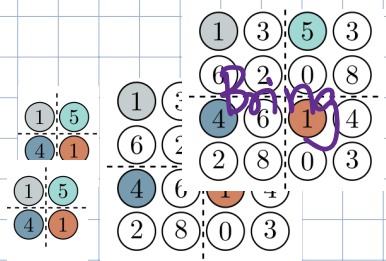
Fundamental tension between getting big picture things and getting details.

U-nets

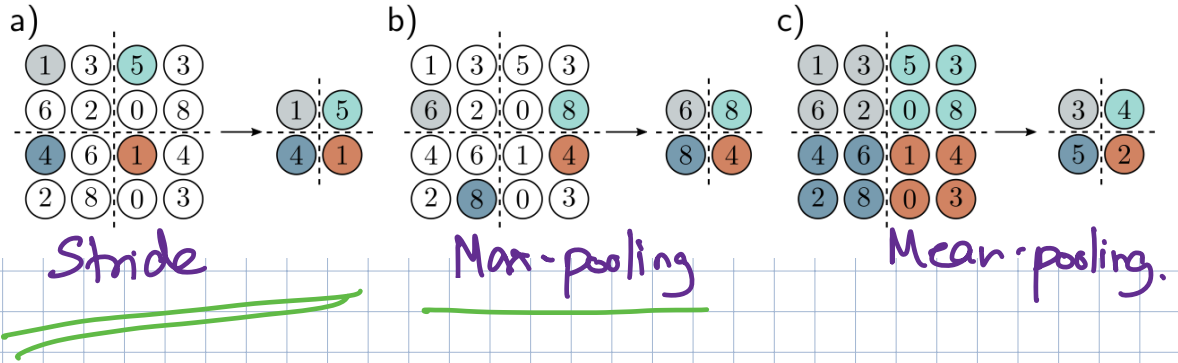


inform this pixel

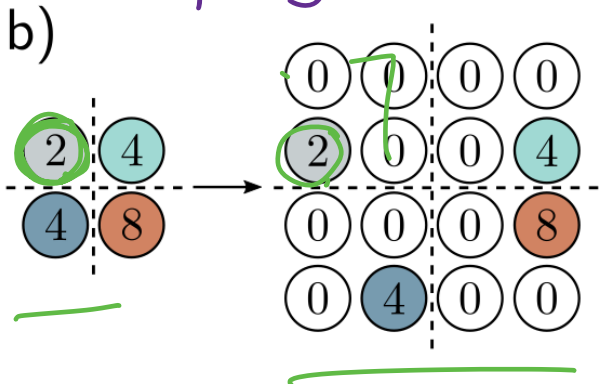
Does not replace block-by-block skip connections.

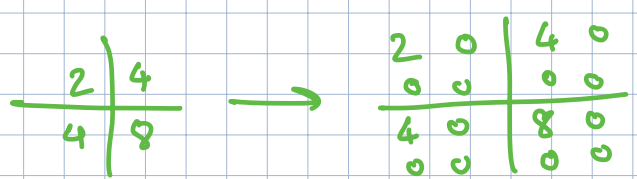


Downsampling

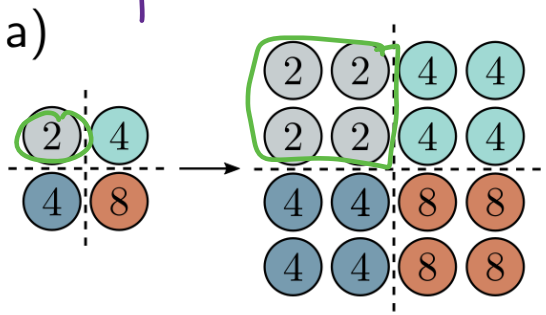


Max unpooling



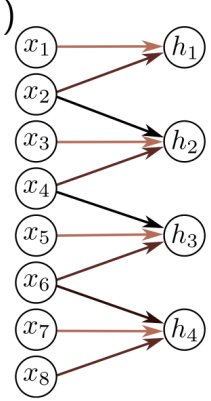


De-duplication.

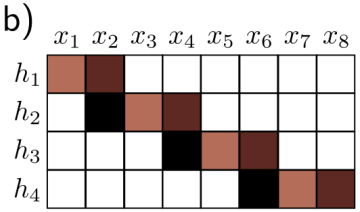


Transpose Convolution

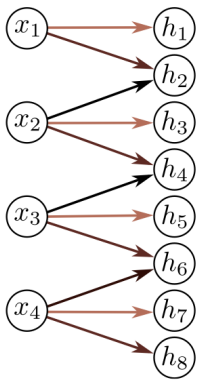
a)



b)



c)



d)

