Advanced Deep Learning

DATA.ML.230

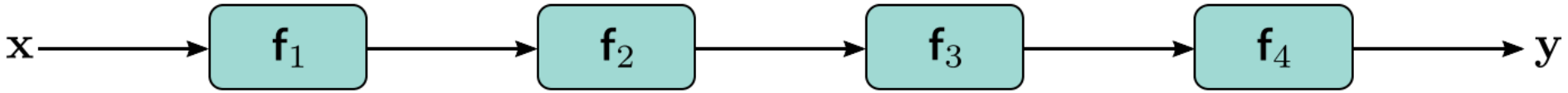
Residual Neural Networks

From: Simon J. D. Prince, Chapter 11 – Understanding Deep Learning, MIT Press (19 May 2025)

Feedforward networks

Until now, we have seen networks which perform sequential processing:

- For a three-layer network:



$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\
\mathbf{h}_2 &= \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\
\mathbf{h}_3 &= \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\
\mathbf{y} &= \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]
\end{aligned}
$$

or $\mathbf{y} = \mathbf{f}_4\Big[\mathbf{f}_3\Big[\mathbf{f}_2\big[\mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1], \boldsymbol{\phi}_2\big], \boldsymbol{\phi}_3\Big], \boldsymbol{\phi}_4\Big]$

Feedforward networks

Limitations of sequential network architectures:

- In general, adding more layers improves performance (increased network capacity).
- However, sometimes performance decreases again as further layers are added.

Why?

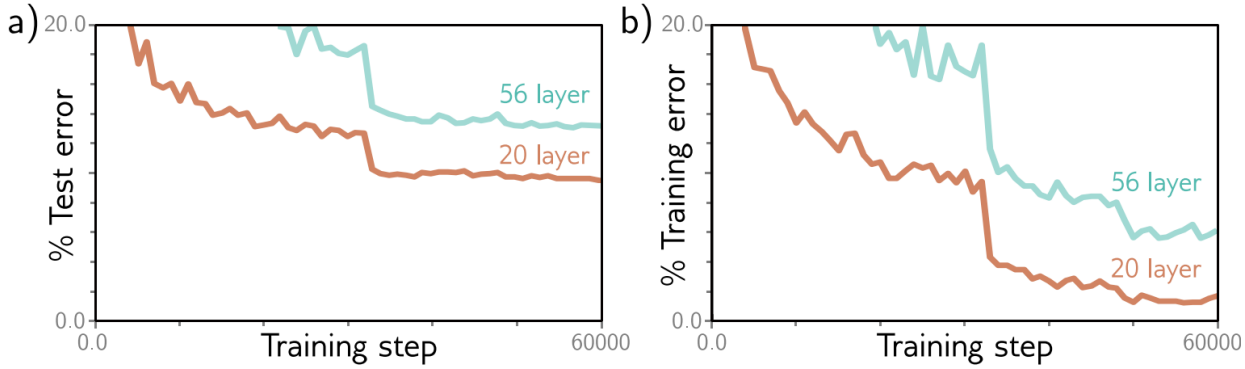


Figure 11.2 Decrease in performance when adding more convolutional layers. a) A 20-layer convolutional network outperforms a 56-layer neural network for image classification on the test set of the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). b) This is also true for the training set, which suggests that the problem relates to training the original network rather than a failure to generalize to new data. Adapted from He et al. (2016a).

Feedforward networks

Limitations of sequential network architectures:

- In general, adding more layers improves performance (increased network capacity).
- However, sometimes performance decreases again as further layers are added.

This is not completely understood

One possible explanation:

The loss gradients change unpredictably in early network layers.

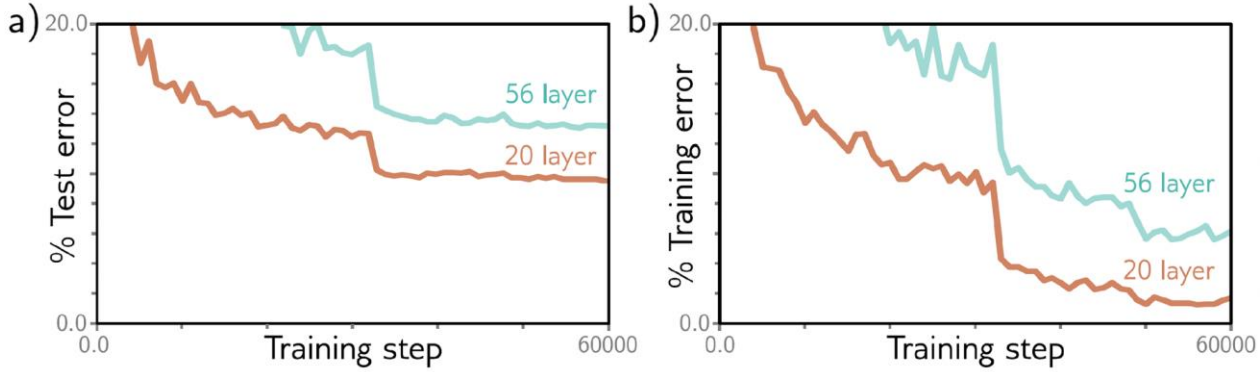


Figure 11.2 Decrease in performance when adding more convolutional layers. a) A 20-layer convolutional network outperforms a 56-layer neural network for image classification on the test set of the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). b) This is also true for the training set, which suggests that the problem relates to training the original network rather than a failure to generalize to new data. Adapted from He et al. (2016a).

Shattered gradients phenomenon

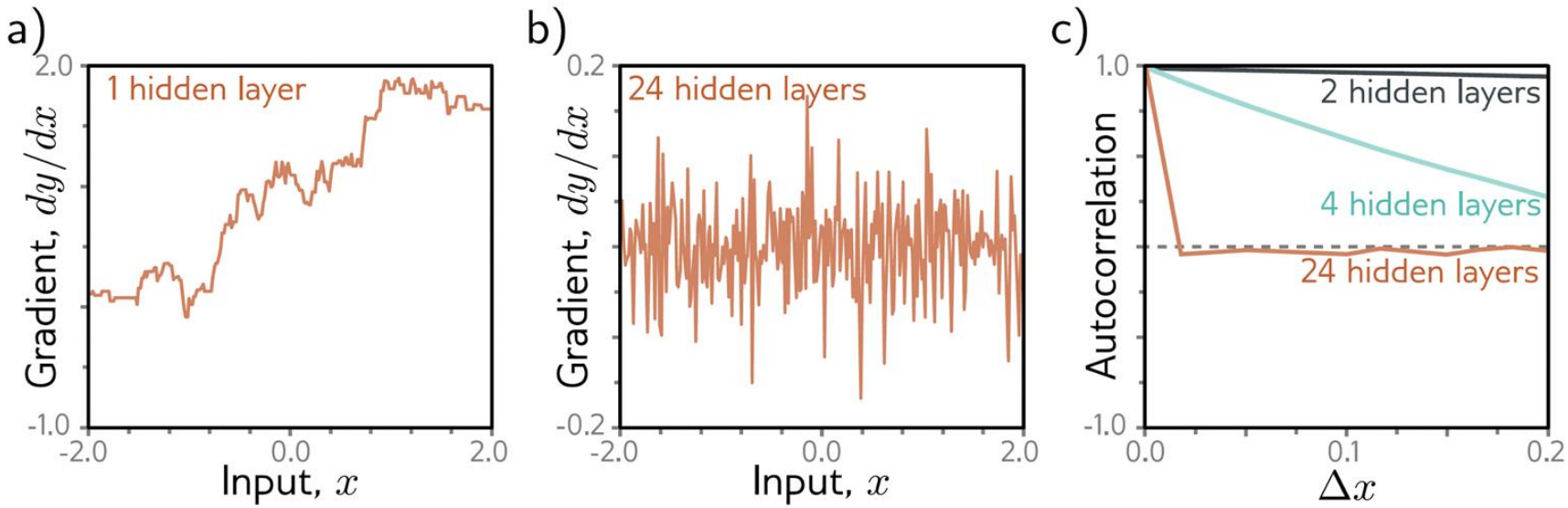


Figure 11.3 Shattered gradients. a) Consider a shallow network with 200 hidden units and Glorot initialization (He initialization without the factor of two) for both the weights and biases. The gradient $\partial y/\partial x$ of the scalar network output y with respect to the scalar input x changes 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).

Shattered gradients phenomenon

Changes in early network layers modify the output in a complex way

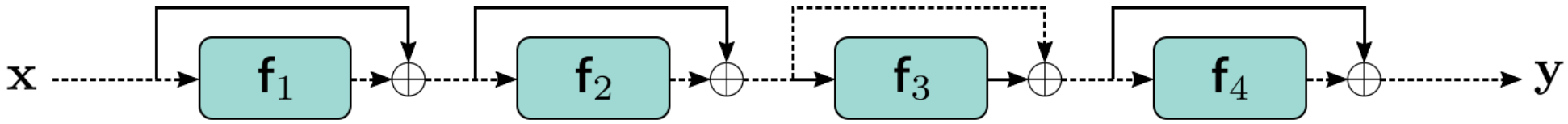
- For the three-layer network:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{f}_1} = \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}$$

- Changing the parameters of layer 1 (determine the output $\mathbf{f}_1$)
  - *All derivatives* are evaluated at different locations
  - The updated gradient may be completely different
  - The loss function becomes badly behaved.

Residual or skip connections

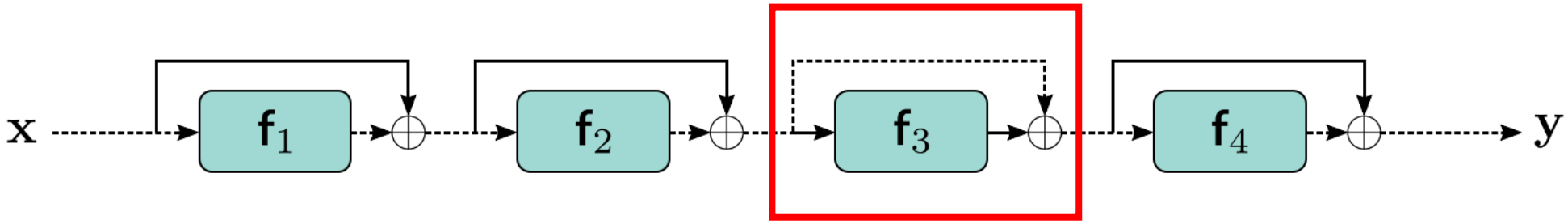
- Add new branches in the computational path



$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]
\end{aligned}
$$

Residual or skip connections

- Add new branches in the computational path

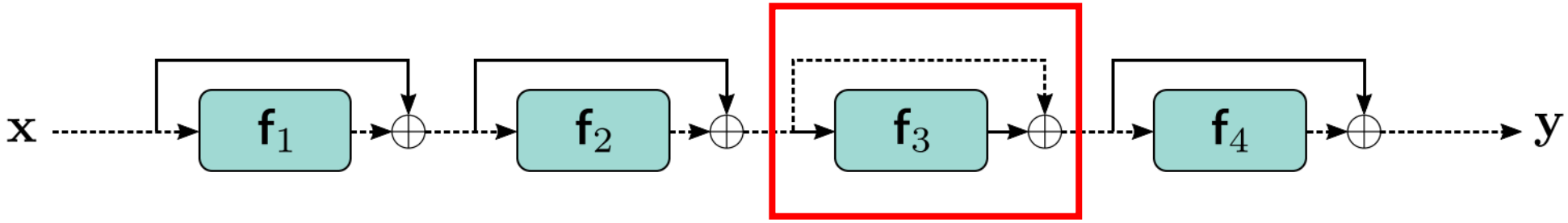


$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]
\end{aligned}
$$

Residual block:
Can be formed by
one or more layers

Residual or skip connections

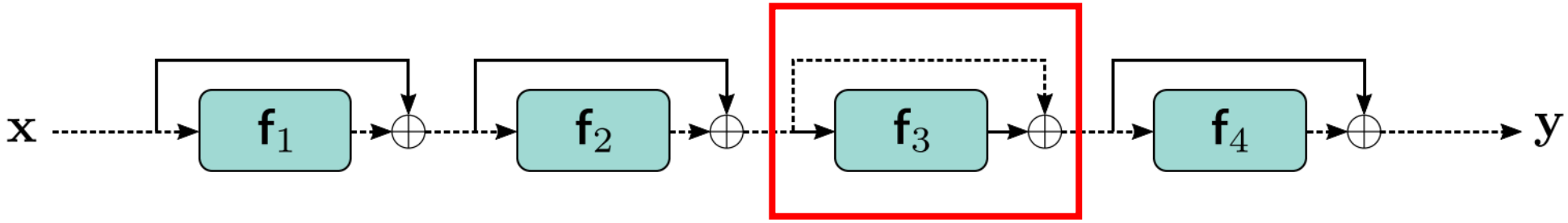
- Add new branches in the computational path



$$
\begin{aligned}
\mathbf{y} = \mathbf{x} &+ \mathbf{f}_1[\mathbf{x}] \\
&+ \mathbf{f}_2[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]] \\
&+ \mathbf{f}_3[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]]] \\
&+ \mathbf{f}_4[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]] + \mathbf{f}_3[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]]]]
\end{aligned}
$$

Residual or skip connections

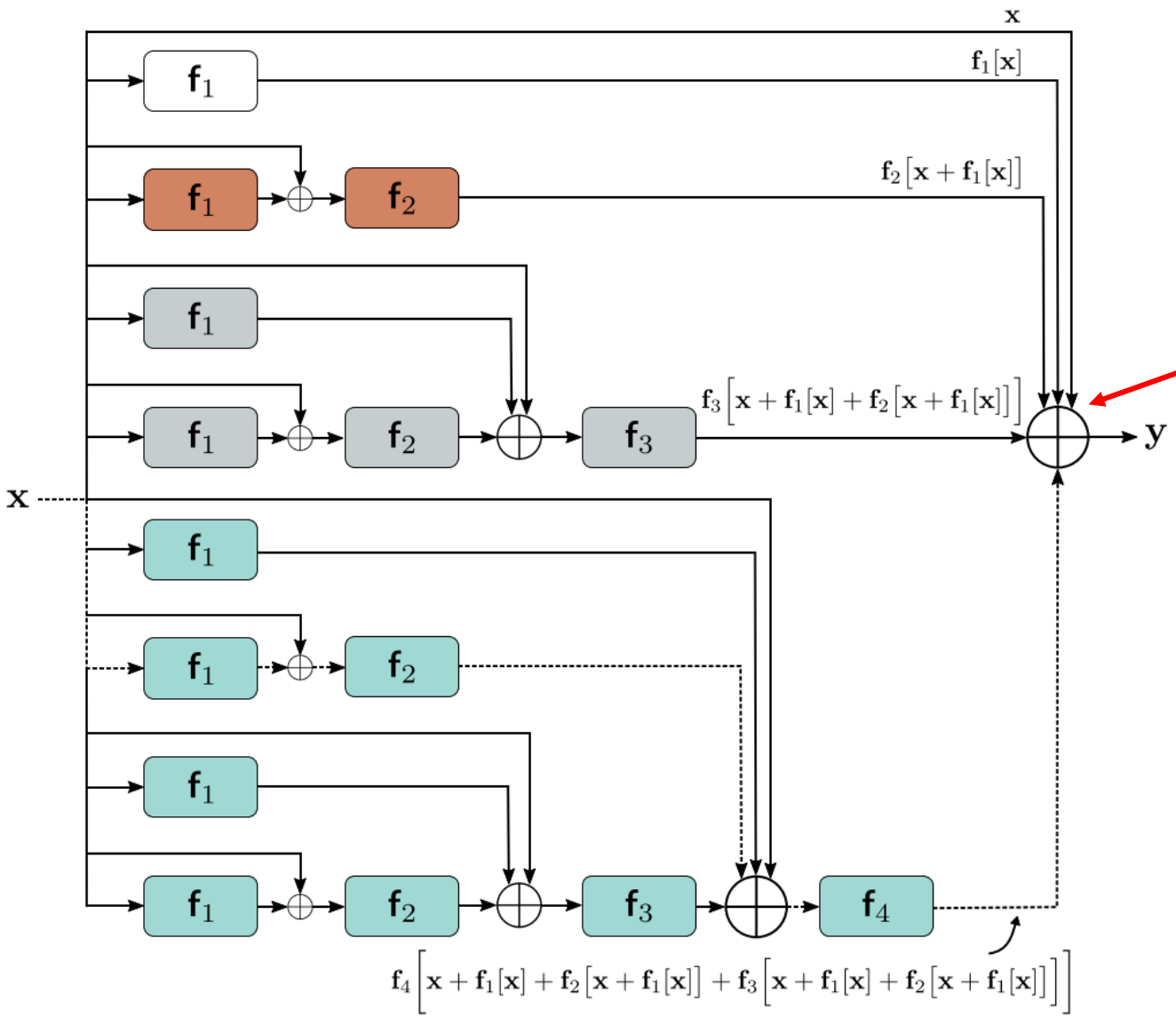
- Add new branches in the computational path



$$
\begin{aligned}
\mathbf{y} = \mathbf{x} &+ \mathbf{f}_1[\mathbf{x}] \\
&+ \mathbf{f}_2\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\right] \\
&+ \mathbf{f}_3\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\right]\right] \\
&+ \mathbf{f}_4\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\right] + \mathbf{f}_3\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}] + \mathbf{f}_2\left[\mathbf{x} + \mathbf{f}_1[\mathbf{x}]\right]\right]\right]
\end{aligned}
$$

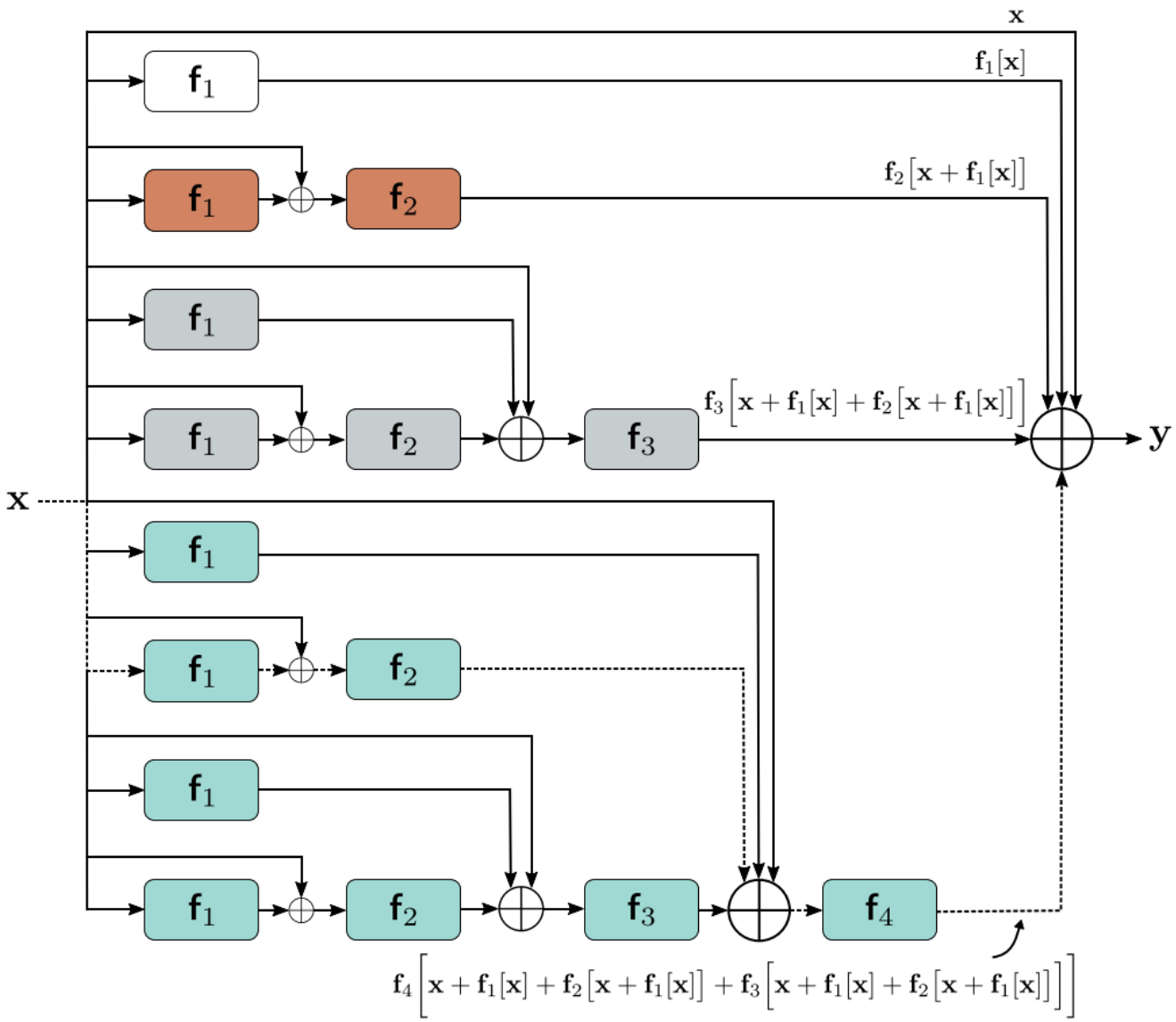
Identity +
4 smaller networks

Residual or skip connections



Residual connections turn the original network to an ensemble of smaller networks

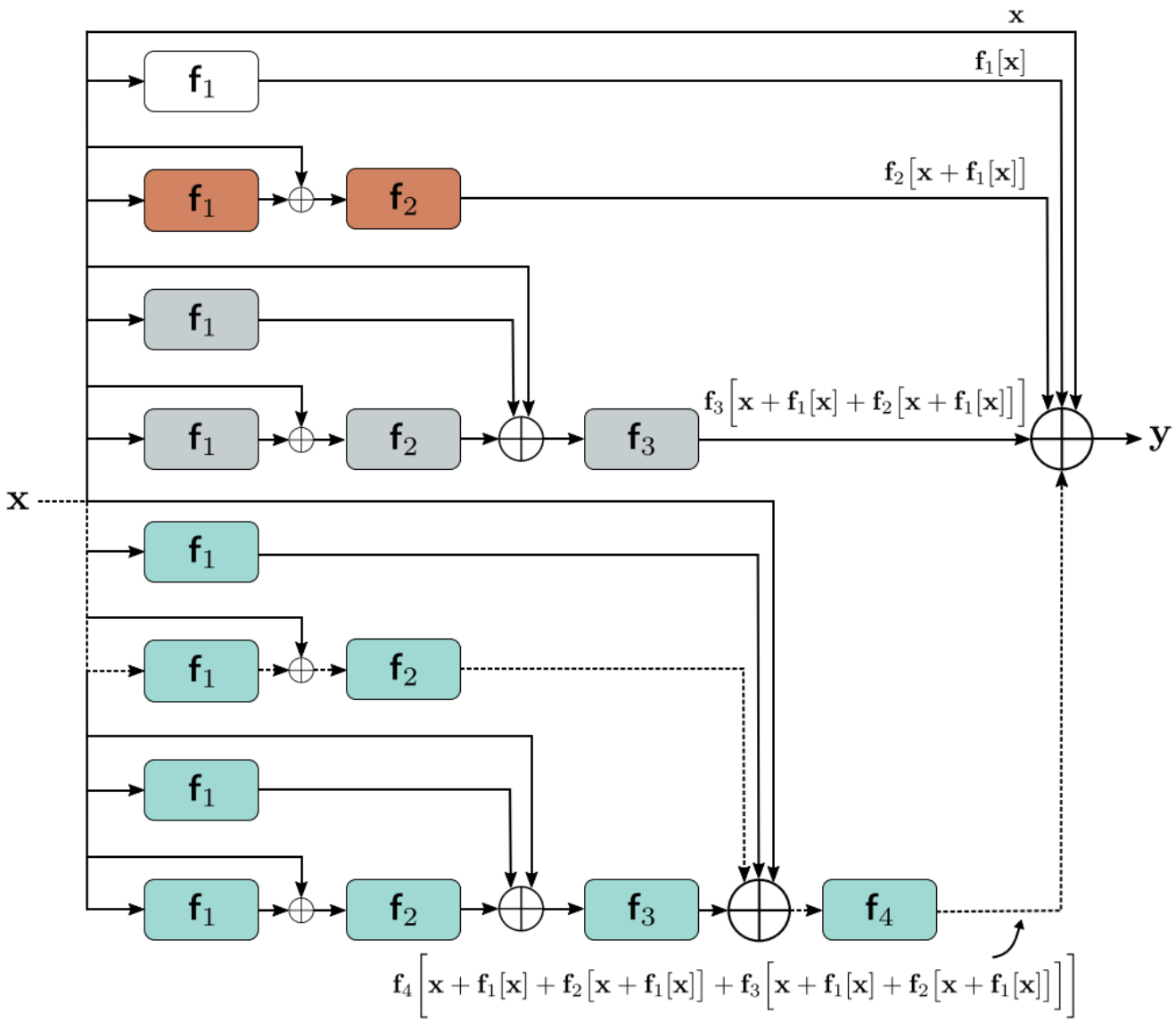
Residual or skip connections



16 paths between input and output.
E.g.:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{f}_1} = \mathbf{I} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} + \left(\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\right) + \left(\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_2} + \frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}\right)$$

Residual or skip connections

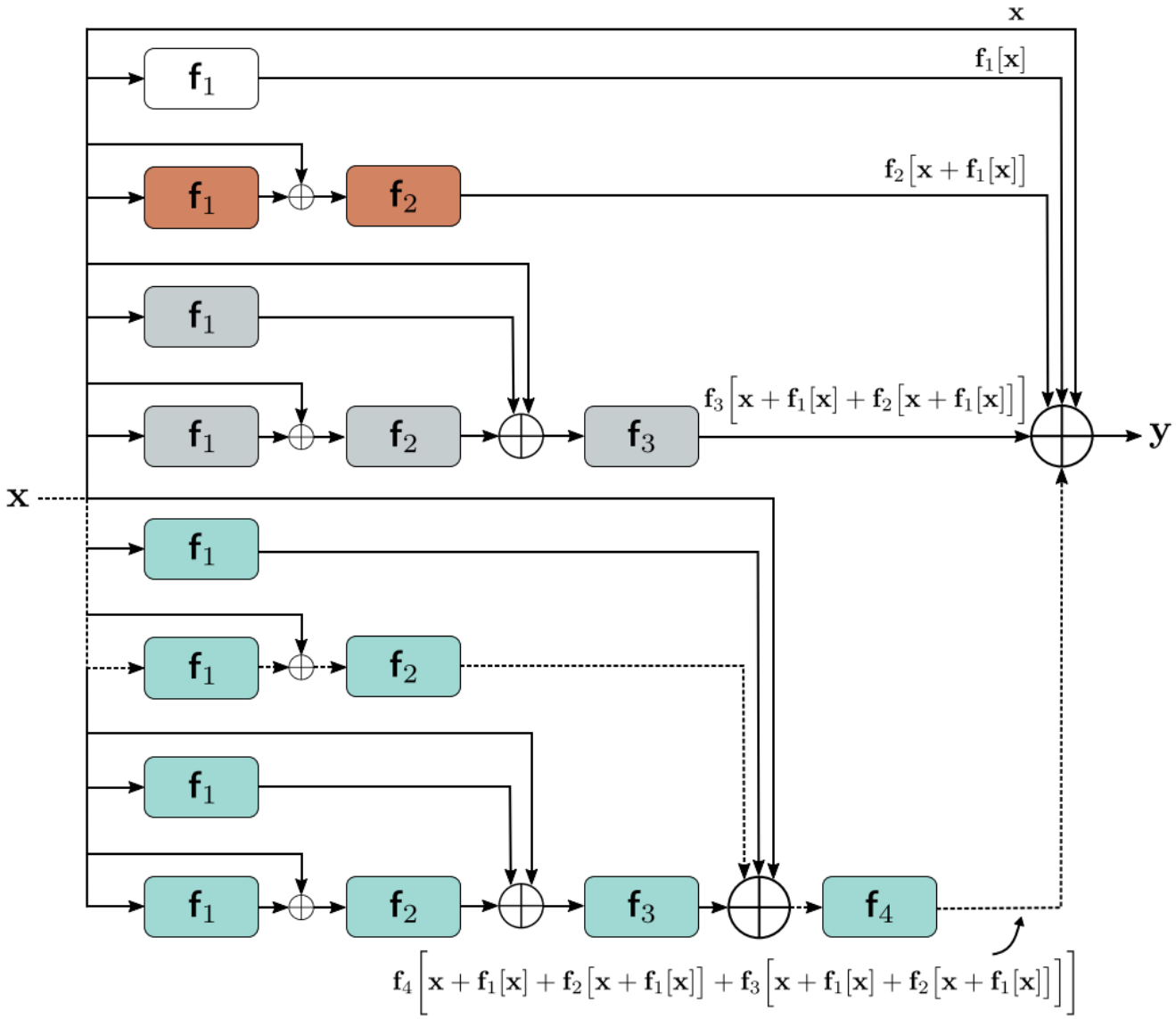


16 paths between input and output.
E.g.:

$$\frac{\partial y}{\partial f_1} = I + \frac{\partial f_2}{\partial f_1} + \left(\frac{\partial f_3}{\partial f_1} + \frac{\partial f_2}{\partial f_1}\frac{\partial f_3}{\partial f_2}\right) + \left(\frac{\partial f_4}{\partial f_1} + \frac{\partial f_2}{\partial f_1}\frac{\partial f_4}{\partial f_2} + \frac{\partial f_3}{\partial f_1}\frac{\partial f_4}{\partial f_3} + \frac{\partial f_2}{\partial f_1}\frac{\partial f_3}{\partial f_2}\frac{\partial f_4}{\partial f_3}\right)$$

Changes in the first layer contribute directly to changes in the output

Residual or skip connections

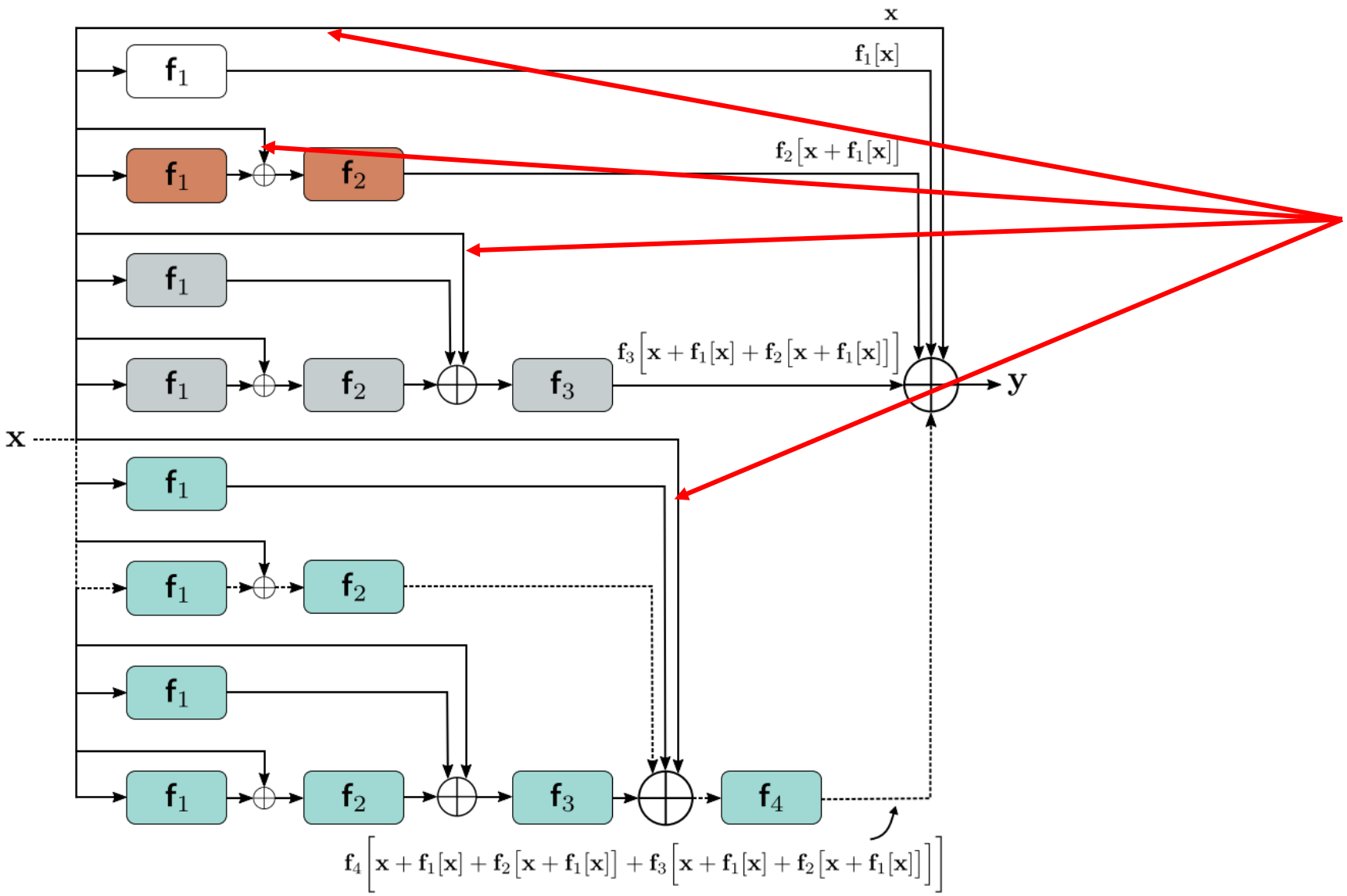


16 paths between input and output.
E.g.:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{f}_1} = \mathbf{I} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} + \left(\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\right) + \left(\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_1} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_2} + \frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3} + \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1}\frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2}\frac{\partial \mathbf{f}_4}{\partial \mathbf{f}_3}\right)$$

The first layer directly contributes to all changes to other layers

Residual or skip connections



Each smaller network has a direct computational path to the input

Networks with residual connections suffer less from shattered gradients

Residual or skip connections

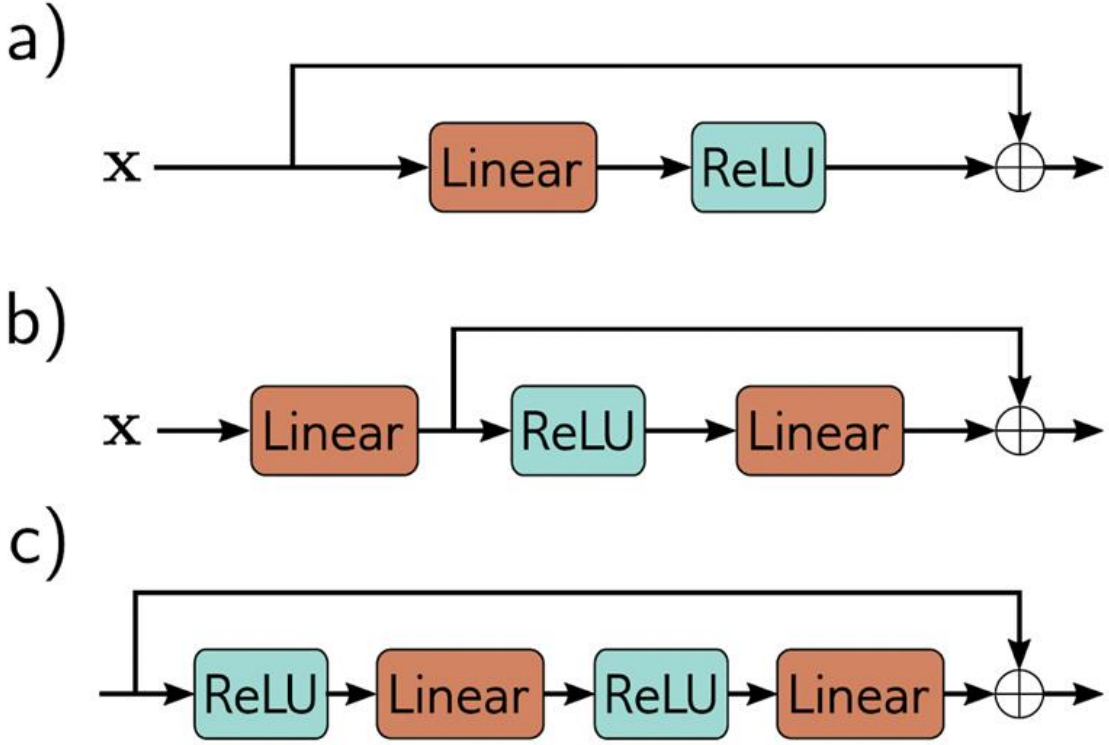


Figure 11.5 Order of operations in residual blocks. a) The usual order of linear transformation or convolution followed by a ReLU nonlinearity means that each residual block can only add non-negative quantities. b) With the reverse order, both positive and negative quantities can be added. However, we 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.

Residual or skip connections (Initialization)

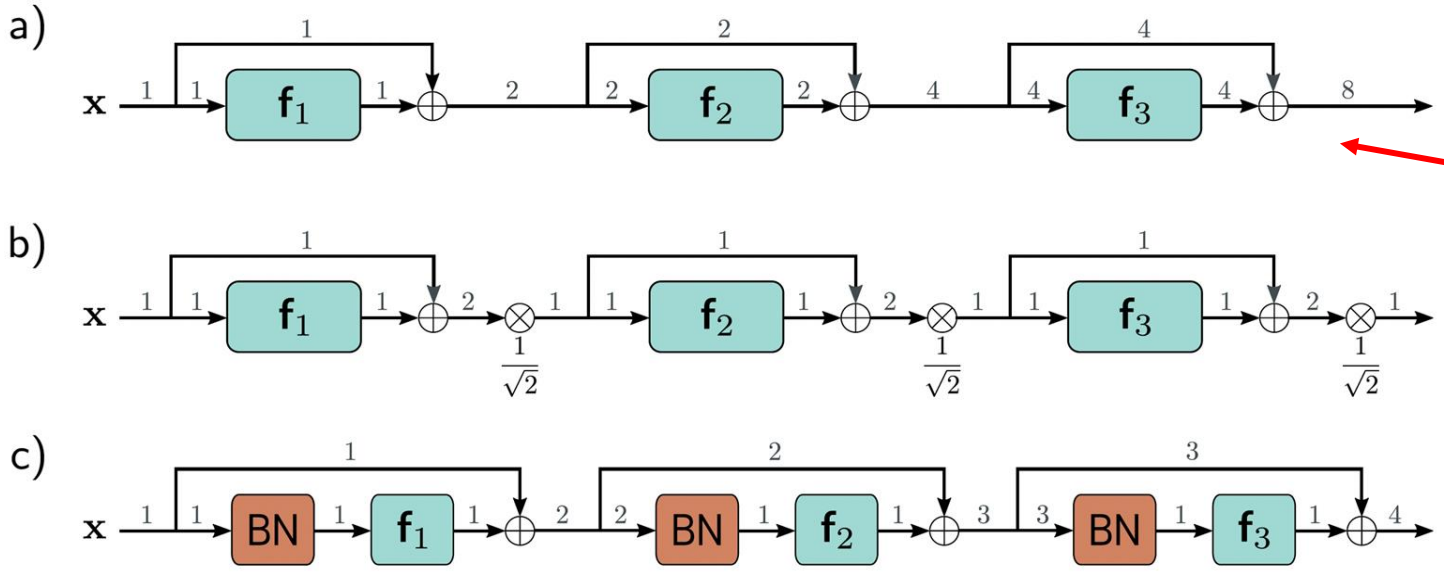


Figure 11.6 Variance in residual networks. a) He initialization ensures that the expected variance remains unchanged after a linear plus ReLU layer $\mathbf{f}_k$. Unfortunately, in residual networks, the input of each block is added back to the output, so the variance doubles at each layer (gray numbers indicate variance) and grows exponentially. b) One approach would be to rescale the signal by $1/\sqrt{2}$ between each residual block. c) A second method uses batch normalization (BN) as the first step in the residual block and initializes the associated offset δ to zero and scale γ to one. This transforms the input to each layer to have unit variance, and with He initialization, the output variance will also be one. Now the variance increases linearly with the number of residual blocks. A side-effect is that, at initialization, later network layers are dominated by the residual connection and are hence close to computing the identity.

He initialization (assumes ReLU)

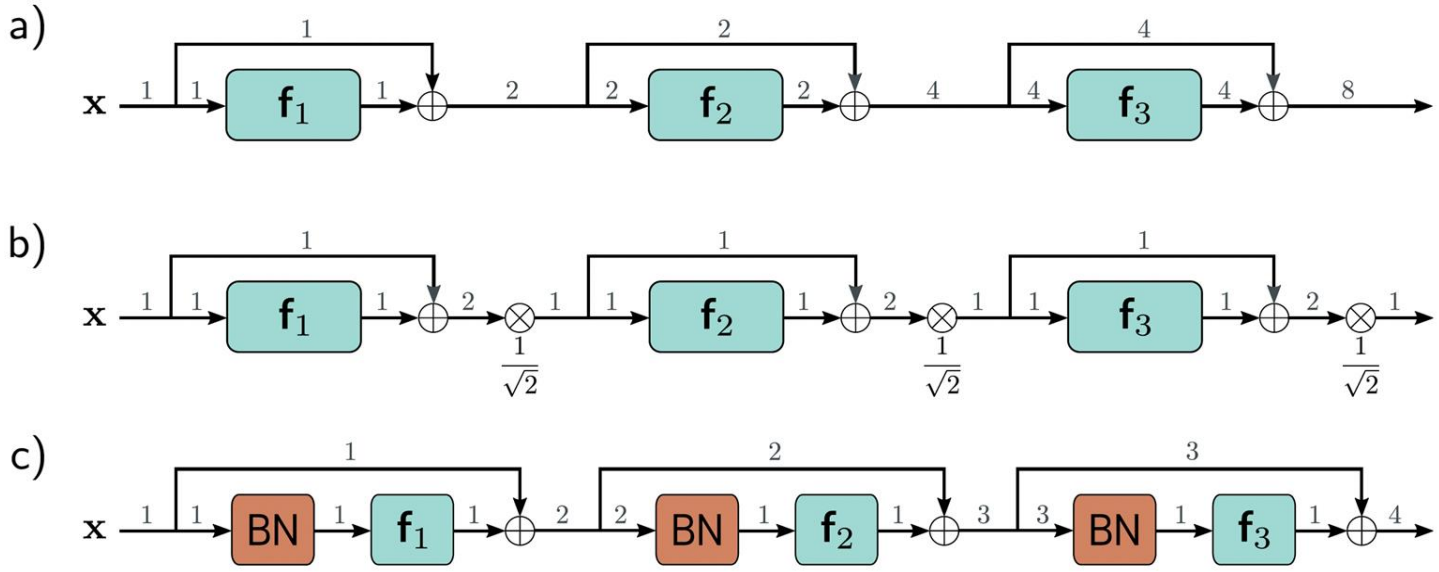
- Forward pass: want the variance of hidden unit activations in layer k+1 to be the same as variance of activations in layer k:

$$\sigma_\Omega^2 = \frac{2}{D_h}$$ ← Number of units at layer k

- Backward pass: want the variance of gradients at layer k to be the same as variance of gradient in layer k+1:

$$\sigma_\Omega^2 = \frac{2}{D_{h'}}$$ ← Number of units at layer k+1

Residual or skip connections (Initialization)



- Use He initialization
- Multiply the combined output of each residual block by $1/\sqrt{2}$

Figure 11.6 Variance in residual networks. a) He initialization ensures that the expected variance remains unchanged after a linear plus ReLU layer $\mathbf{f}_k$. Unfortunately, in residual networks, the input of each block is added back to the output, so the variance doubles at each layer (gray numbers indicate variance) and grows exponentially. b) One approach would be to rescale the signal by $1/\sqrt{2}$ between each residual block. c) A second method uses batch normalization (BN) as the first step in the residual block and initializes the associated offset δ to zero and scale γ to one. This transforms the input to each layer to have unit variance, and with He initialization, the output variance will also be one. Now the variance increases linearly with the number of residual blocks. A side-effect is that, at initialization, later network layers are dominated by the residual connection and are hence close to computing the identity.

Residual or skip connections (Initialization)

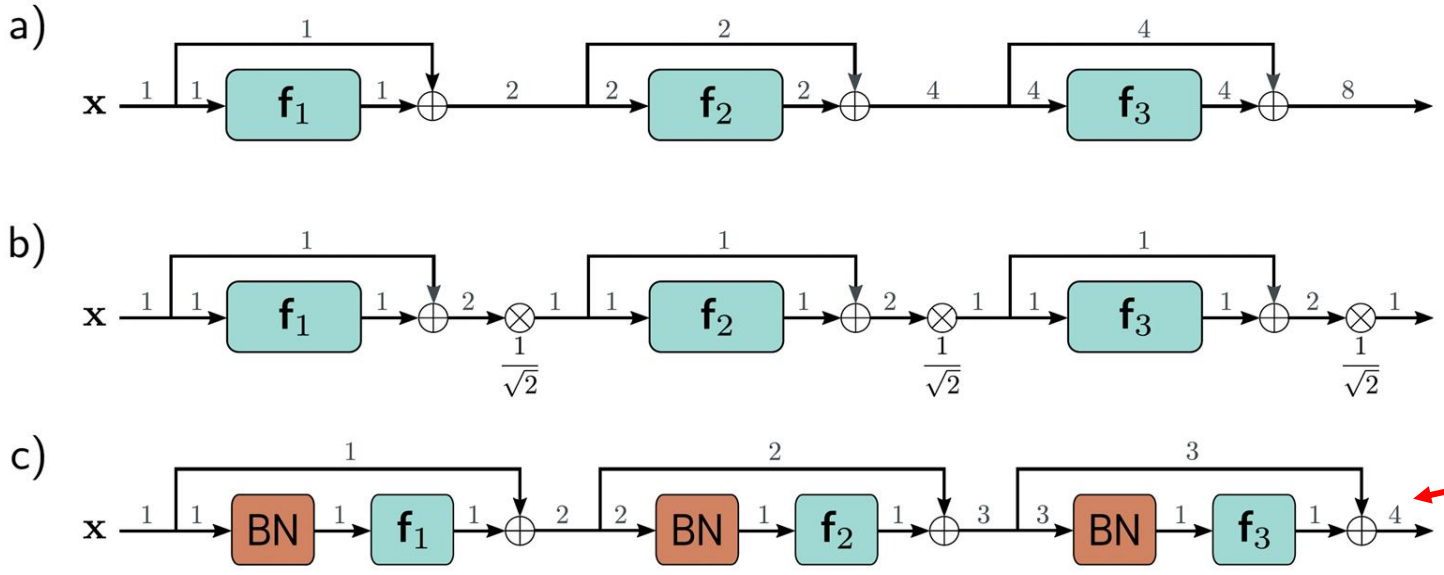


Figure 11.6 Variance in residual networks. a) He initialization ensures that the expected variance remains unchanged after a linear plus ReLU layer $\mathbf{f}_k$. Unfortunately, in residual networks, the input of each block is added back to the output, so the variance doubles at each layer (gray numbers indicate variance) and grows exponentially. b) One approach would be to rescale the signal by $1/\sqrt{2}$ between each residual block. c) A second method uses batch normalization (BN) as the first step in the residual block and initializes the associated offset δ to zero and scale γ to one. This transforms the input to each layer to have unit variance, and with He initialization, the output variance will also be one. Now the variance increases linearly with the number of residual blocks. A side-effect is that, at initialization, later network layers are dominated by the residual connection and are hence close to computing the identity.

Batch normalization:

- Stable forward propagation
- Higher learning rates
- Acts as a form of regularization (injects noise to the training updates)

Batch normalization

Shits and rescales each activation h so that its mean and variance within the batch $\mathcal{B}$ become values learned during training:

- Compute empirical mean and standard deviation:

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

- Use these to standardize the batch activations:

$$h_i \leftarrow \frac{h_i - m_h}{s_h + \epsilon} \qquad \forall i \in \mathcal{B}$$

- Scale and shift the normalized variables

$$h_i \leftarrow \gamma h_i + \delta \qquad \forall i \in \mathcal{B}$$

Batch normalization

Shits and rescales each activation h so that its mean and variance within the batch $\mathcal{B}$ become values learned during training:

- Compute empirical mean and standard deviation:

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

- Use these to standardize the batch activations:

$$h_i \leftarrow \frac{h_i - m_h}{s_h + \epsilon} \qquad \forall i \in \mathcal{B}$$

- Scale and shift the normalized variables

$$h_i \leftarrow \gamma h_i + \delta \qquad \forall i \in \mathcal{B}$$

Initialized to
δ = 0 and γ = 1
Tuned during training

Batch normalization

Shits and rescales each activation h so that its mean and variance within the batch $\mathcal{B}$ become values learned during training:

- Compute empirical mean and standard deviation:

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

- Use these to standardize the batch activations:

$$h_i \leftarrow \frac{h_i - m_h}{s_h + \epsilon} \qquad \forall i \in \mathcal{B}$$

For testing, m_h and s_h of the entire training set are used

- Scale and shift the normalized variables

$$h_i \leftarrow \gamma h_i + \delta \qquad \forall i \in \mathcal{B}$$

The learnt γ and δ are used in testing

Normalization schemes

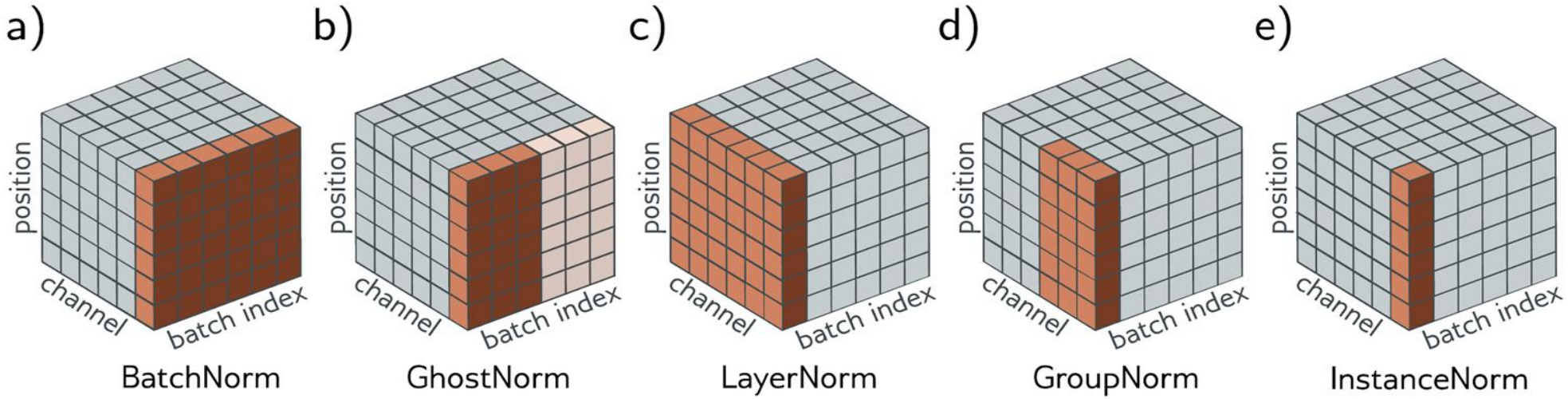
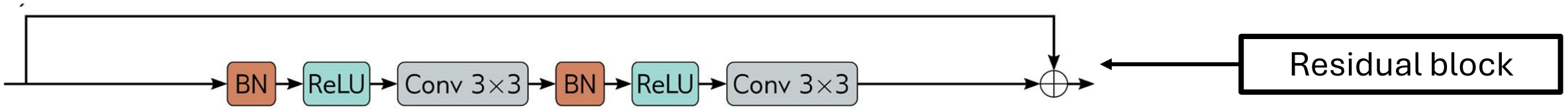
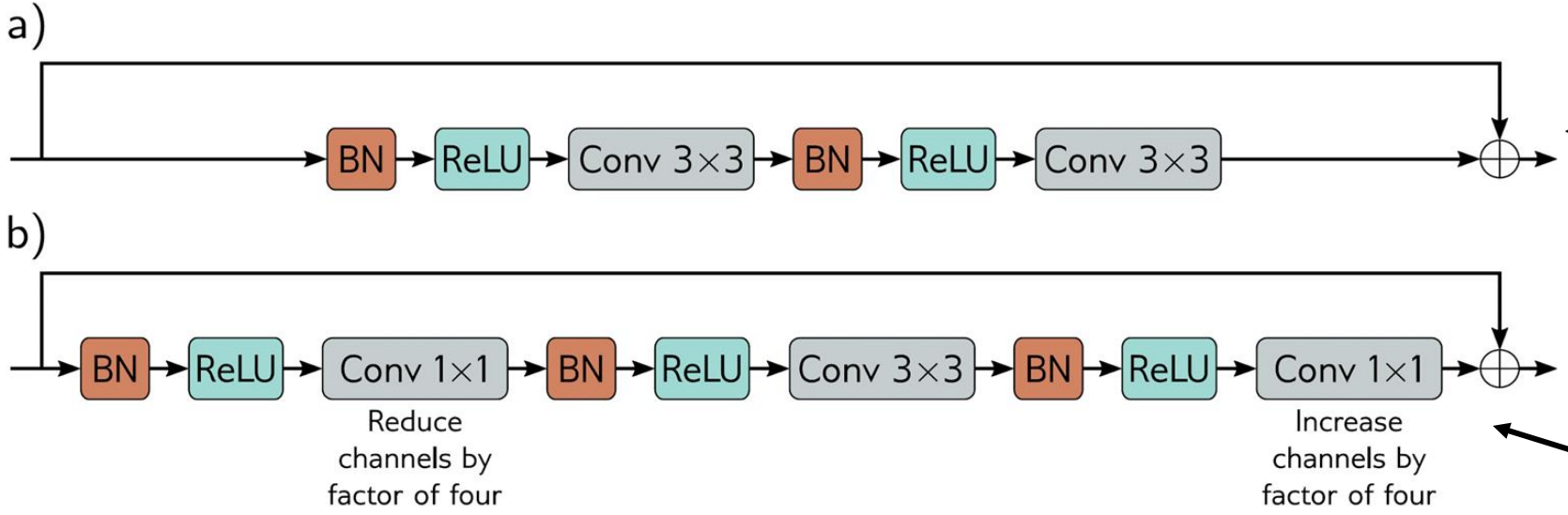


Figure 11.14 Normalization schemes. BatchNorm modifies each channel separately but adjusts each batch member in the same way based on statistics gathered across the batch and spatial position. Ghost BatchNorm computes these statistics from only part of the batch to make them more variable. LayerNorm 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).

Residual architectures: ResNets



Residual architectures: ResNets



Residual block

For very deep networks bottleneck residual blocks make efficient use of parameters

Figure 11.7 ResNet blocks. a) A standard block in the ResNet architecture contains a batch normalization operation, followed by an activation function, and a 3×3 convolutional layer. Then, this sequence is repeated. b). A bottleneck ResNet block still integrates information over a 3×3 region but uses fewer parameters. It contains three convolutions. The first 1×1 convolution reduces the number of channels. The second 3×3 convolution is applied to the smaller representation. A final 1×1 convolution increases the number of channels again so that it can be added back to the input.

Residual architectures: ResNet200

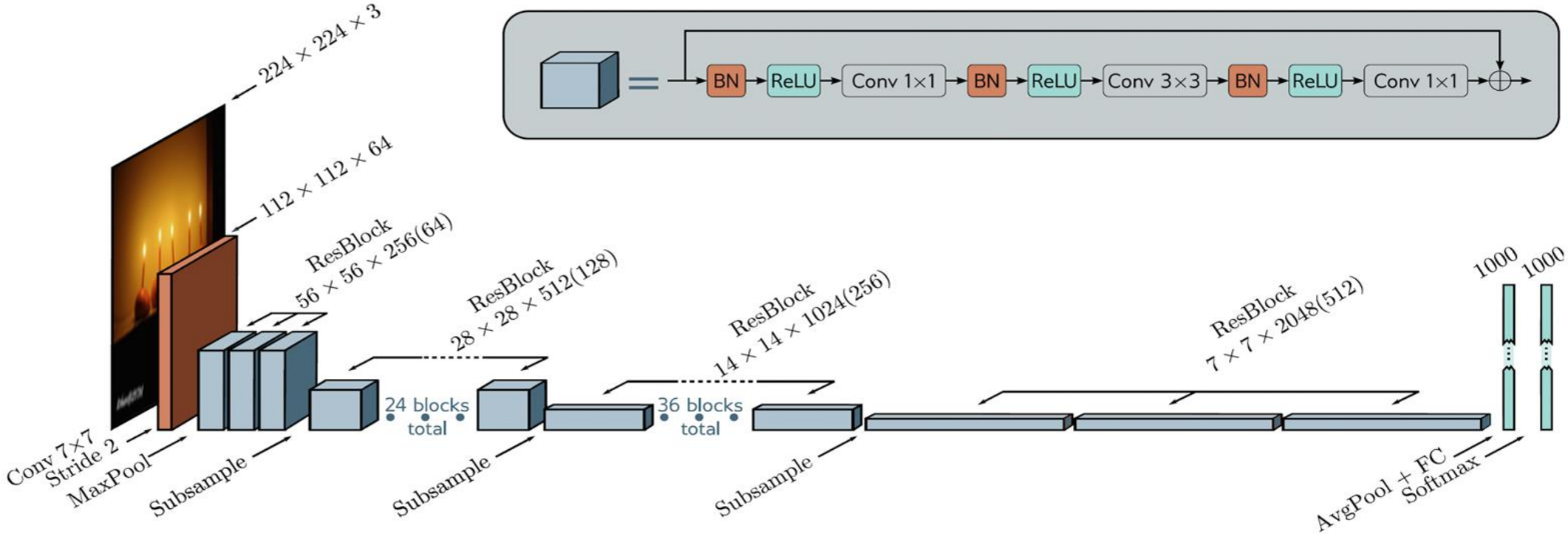


Figure 11.8 ResNet-200 model. A standard 7×7 convolutional layer with stride two is applied, followed by a MaxPool operation. A series of bottleneck residual blocks follow (number in brackets is channels after first 1×1 convolution), with periodic downsampling and accompanying increases in the number of channels. The network concludes with average pooling across all spatial positions and a fully connected layer that maps to pre-softmax activations.

Residual architectures: DenseNet

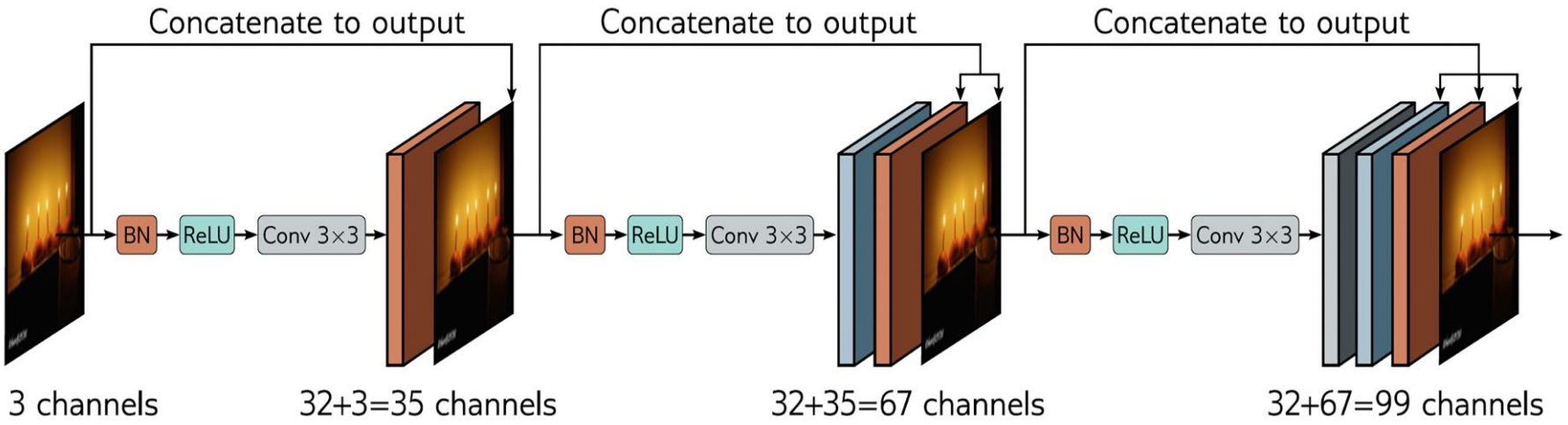


Figure 11.9 DenseNet. This architecture uses residual connections to concatenate the outputs of earlier layers to later ones. Here, the three-channel input image is processed to form a 32-channel representation. The input image is concatenated to this to give a total of 35 channels. This combined representation is processed to create another 32-channel representation, and both earlier representations are concatenated to this to create a total of 67 channels and so on.

Instead of adding input of a block to its output:

- Concatenate the input of a block to its output
- Use a learnable transformation (1x1 convolution) to map the result to lower dimensions

Can outperform ResNets with similar number of parameters

Residual architectures: U-Net

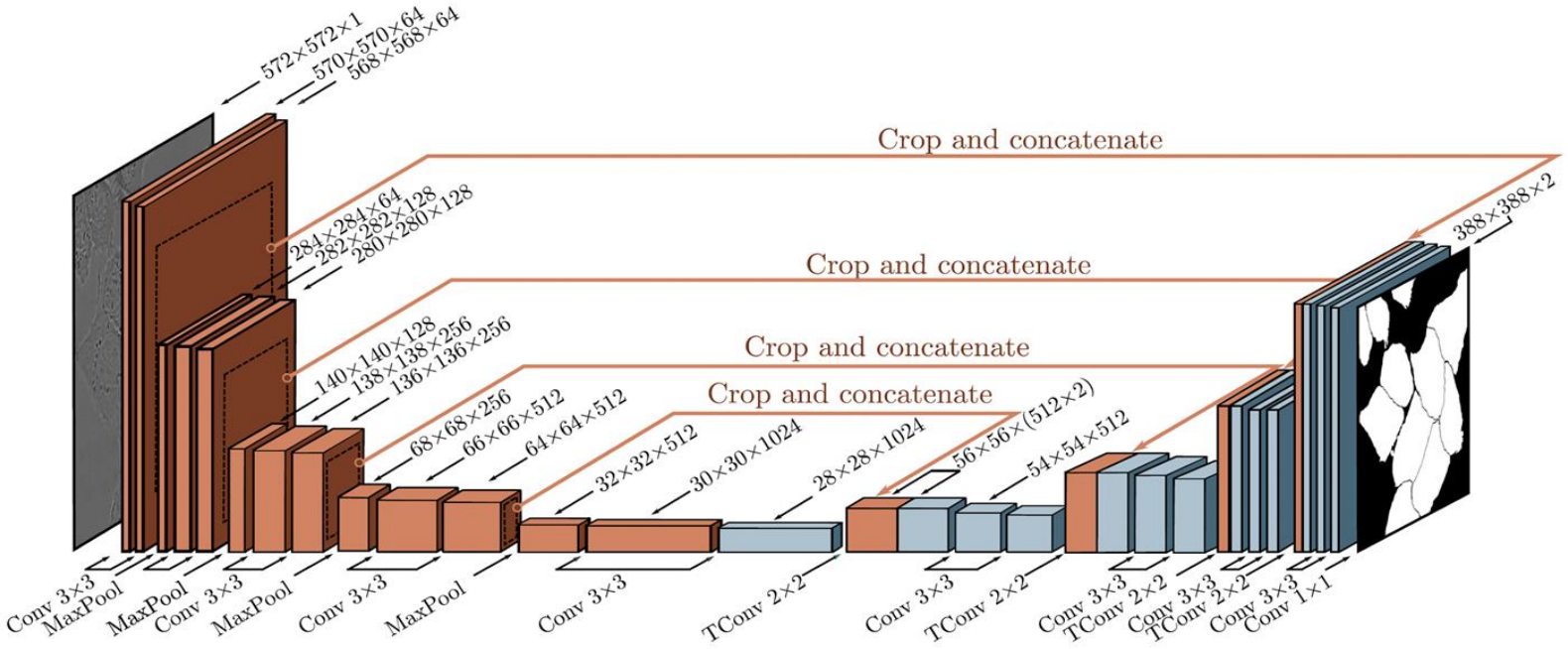


Figure 11.10 U-Net for segmenting HeLa cells. The U-Net has an encoder-decoder structure, in which the representation is downsampled (orange blocks) and then re-upsampled (blue blocks). The encoder uses regular convolutions, and the decoder uses transposed convolutions. Residual connections append the last representation at each scale in the encoder to the first representation at the same scale in the decoder (orange arrows). The original U-Net used "valid" convolutions, so the size decreased slightly with each layer, even without downsampling. Hence, the representations from the encoder were cropped (dashed squares) before appending to the decoder. Adapted from Ronneberger et al. (2015).

Encoder-Decoder architecture where earlier representations are concatenated to later ones.

Used for image-to-image mapping tasks, like image segmentation

Residual architectures: U-Net

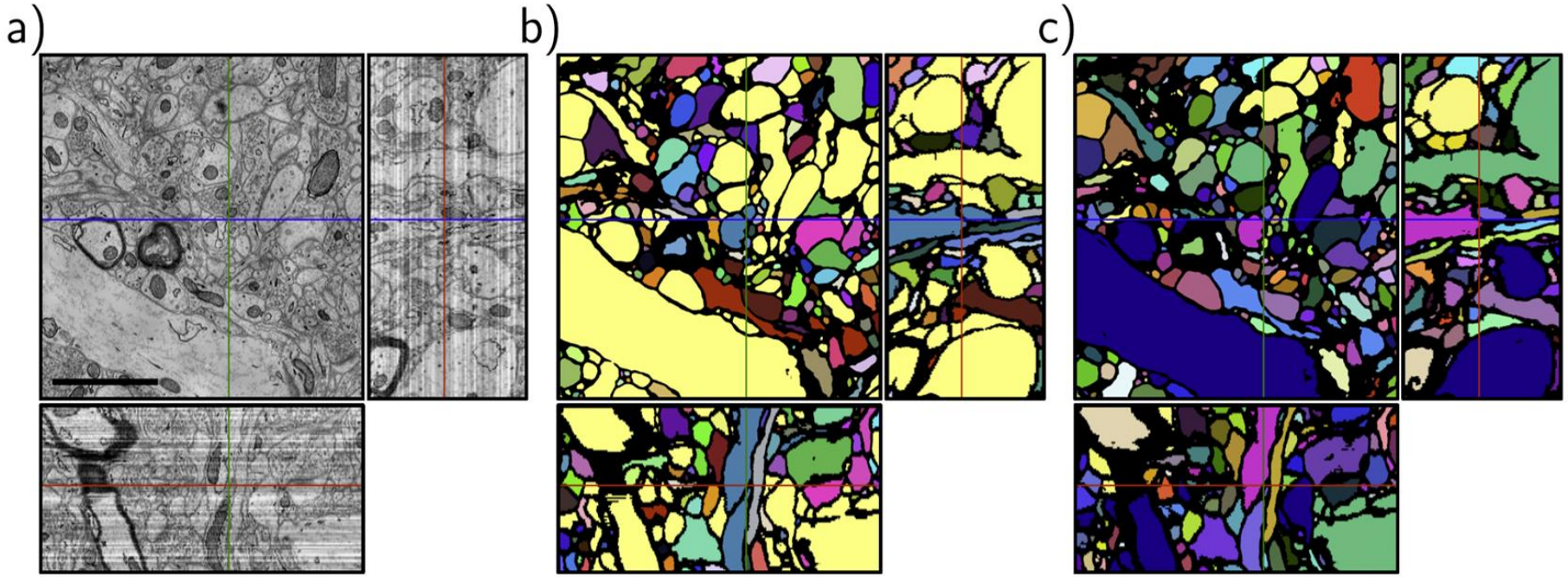
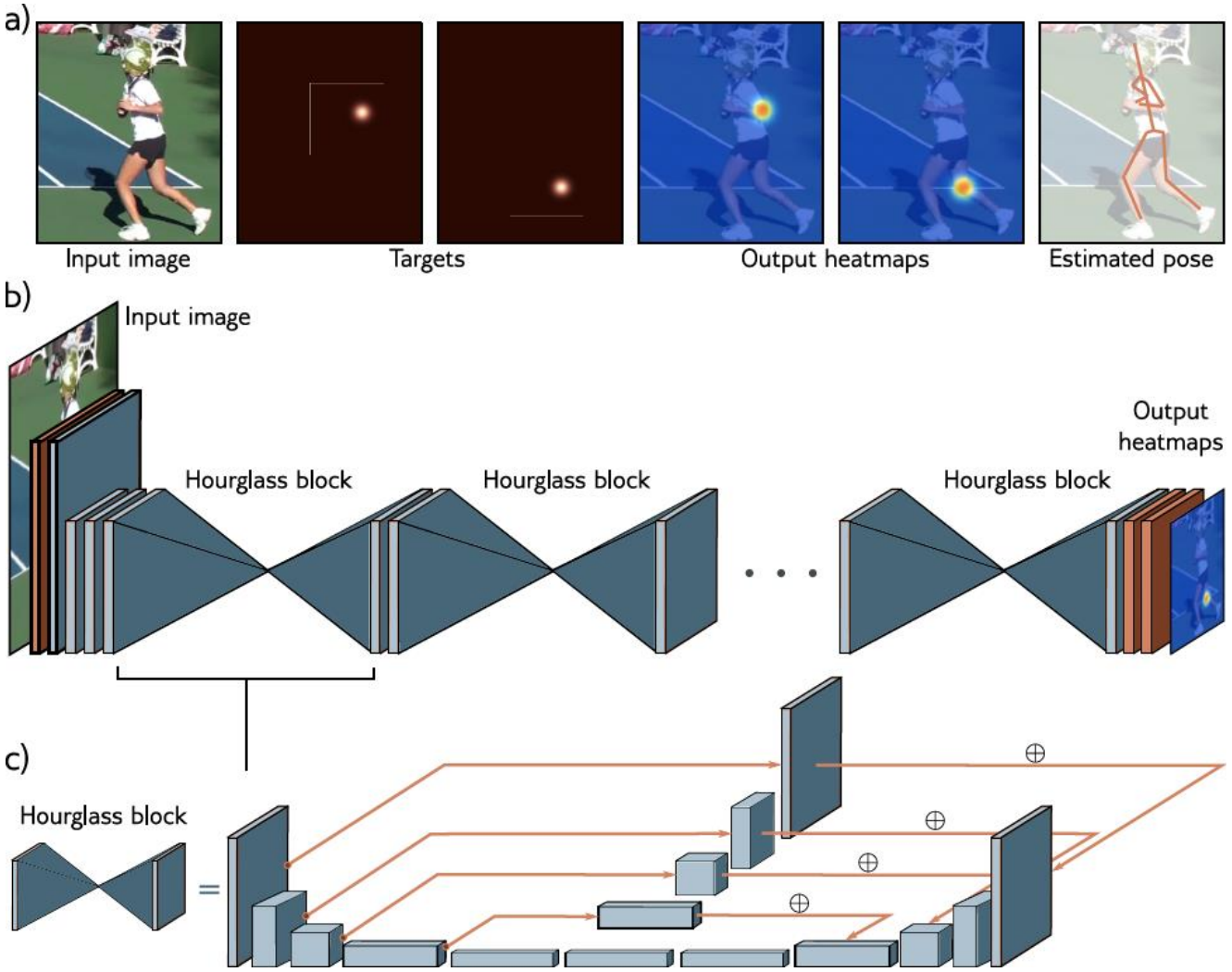


Figure 11.11 Segmentation using U-Net in 3D. a) Three slices through a 3D volume of mouse cortex taken by scanning electron microscope. b) A single U-Net is used to classify voxels as being inside or outside neurites. Connected regions are identified with different colors. c) For a better result, an ensemble of five U-Nets is trained, and a voxel is only classified as belonging to the cell if all five networks agree. Adapted from Falk et al. (2019).

Residual architectures: Hourglas networks



Similar to ResNets but:

- Apply further convolutional layers in the skip connections
- Add the result back to the decoder (no concatenation)

A series of Hourglass models forms a Stacked Hourglass network