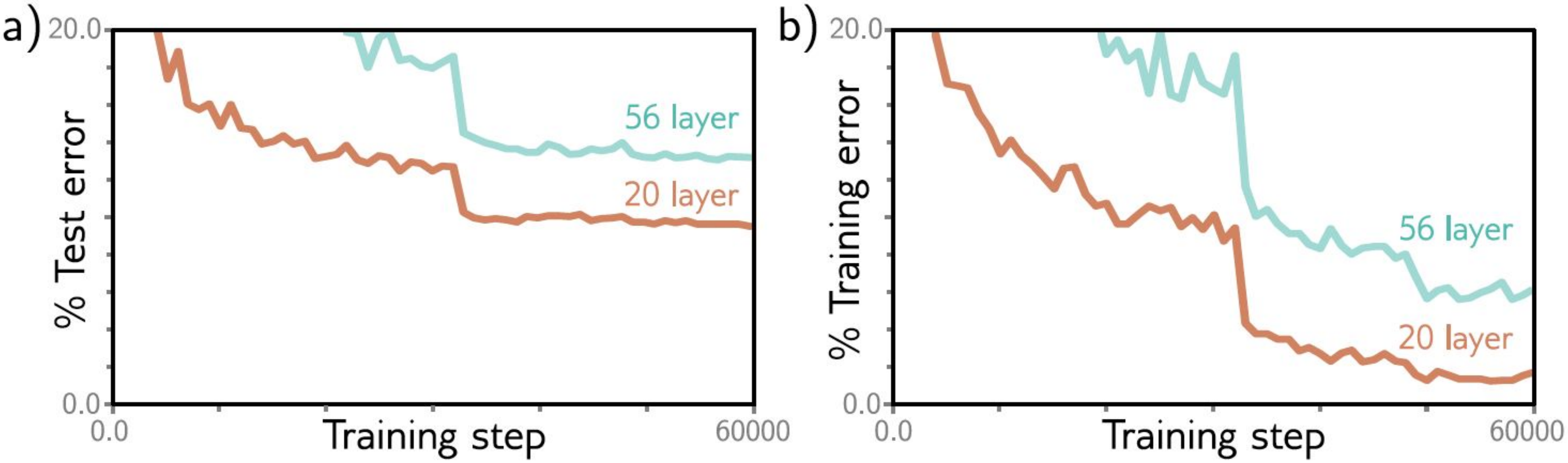
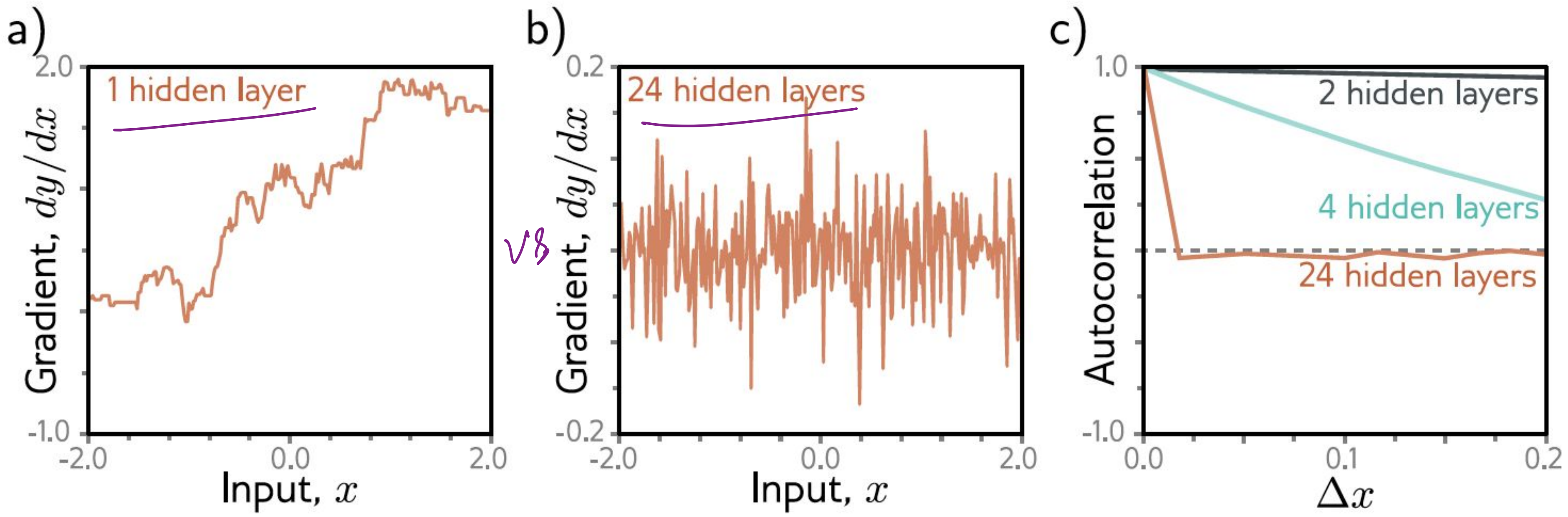
Convolution #2

- 2D Convolution
- Downsampling and upsampling, 1x1 convolution
- Image classification
- Object detection
- Semantic segmentation
- Residual networks
- U-Nets and hourglass networks



If ResNet is the solution what is the question?



Shattered Gradients: (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.*

Regular network:

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

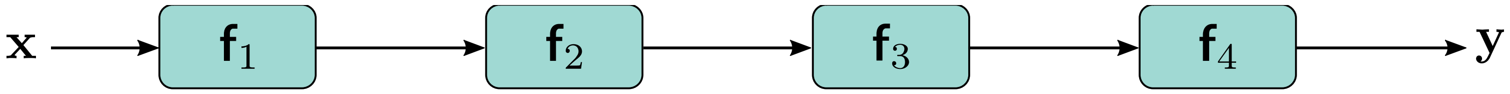
$$\mathbf{x} \rightarrow \boxed{\mathbf{f}_1} \rightarrow \boxed{\mathbf{f}_2} \rightarrow \boxed{\mathbf{f}_3} \rightarrow \boxed{\mathbf{f}_4} \rightarrow \mathbf{y}$$

Shattered Gradients

$$\frac{\partial y}{\partial x} = \cancel{\frac{\partial f}{\partial f_4}} \cdot \frac{\partial f_1}{\partial x} \frac{\partial f_2}{\partial f_1} \frac{\partial f_3}{\partial f_2} \frac{\partial f_4}{\partial f_3} \frac{\partial L}{\partial f_4}$$

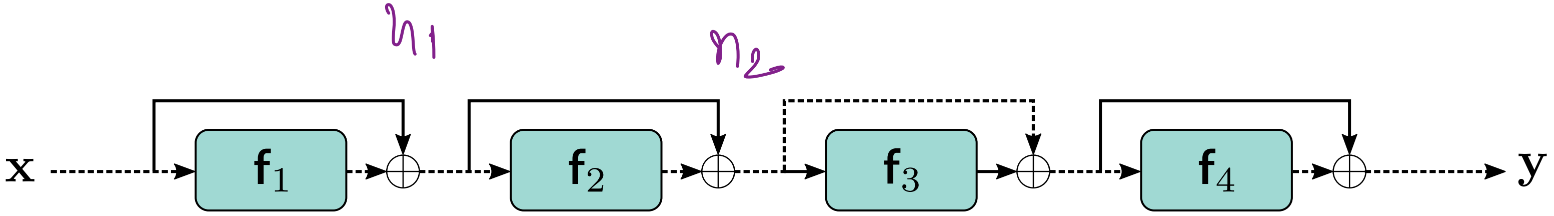
Regular network:

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{f}_1[\mathbf{x}, \phi_1] \\
\mathbf{h}_2 &= \mathbf{f}_2[\mathbf{h}_1, \phi_2] \\
\mathbf{h}_3 &= \mathbf{f}_3[\mathbf{h}_2, \phi_3] \\
\mathbf{y} &= \mathbf{f}_4[\mathbf{h}_3, \phi_4]
\end{aligned}
$$



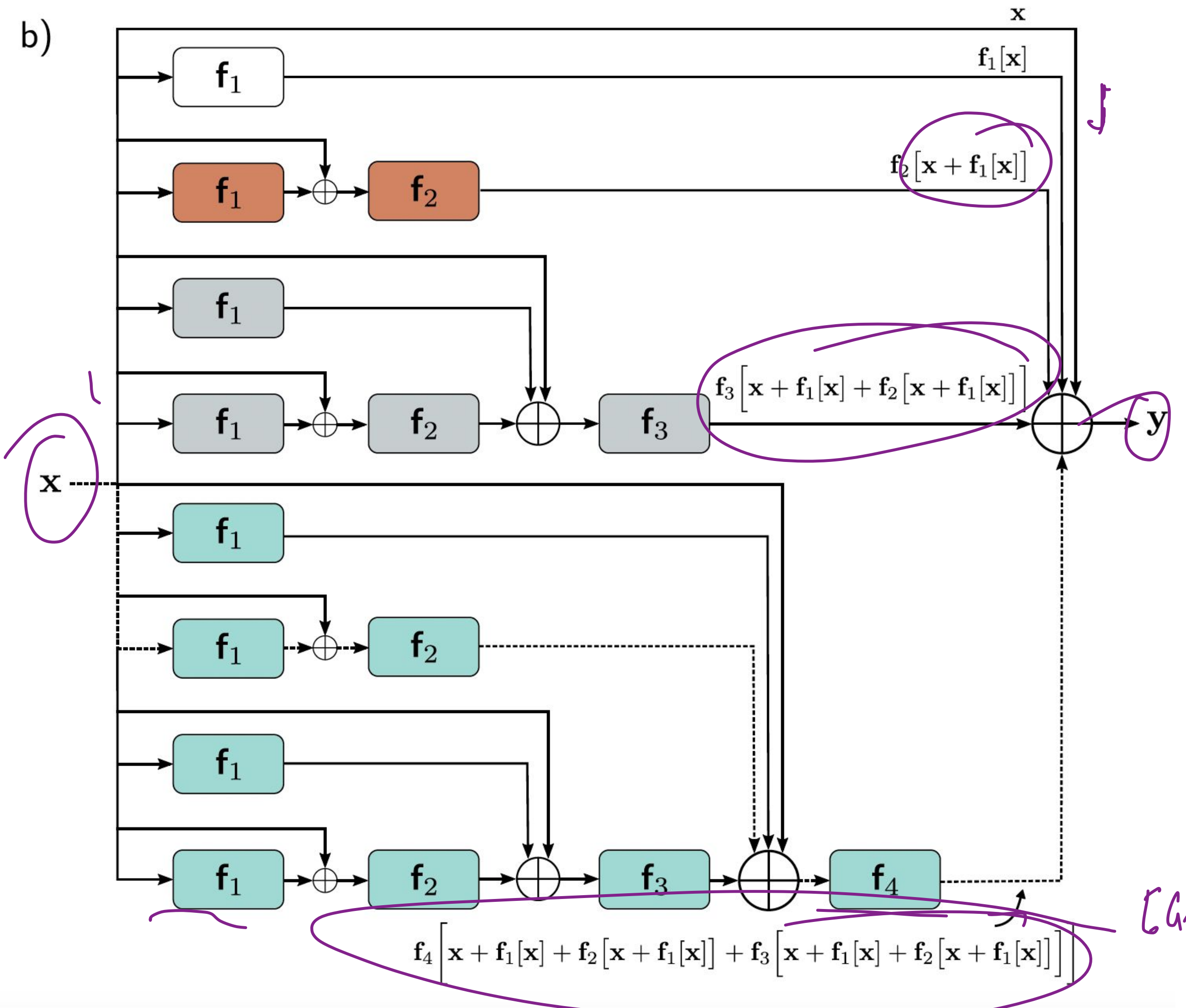
Residual network (2016):

$$
\begin{aligned}
\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \phi_1] \\
\mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \phi_2] \\
\mathbf{h}_3 &= \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \phi_3] \\
\mathbf{y} &= \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \phi_4]
\end{aligned}
$$



$f_2(x+f_1(x)) + f_1(x)$

b)



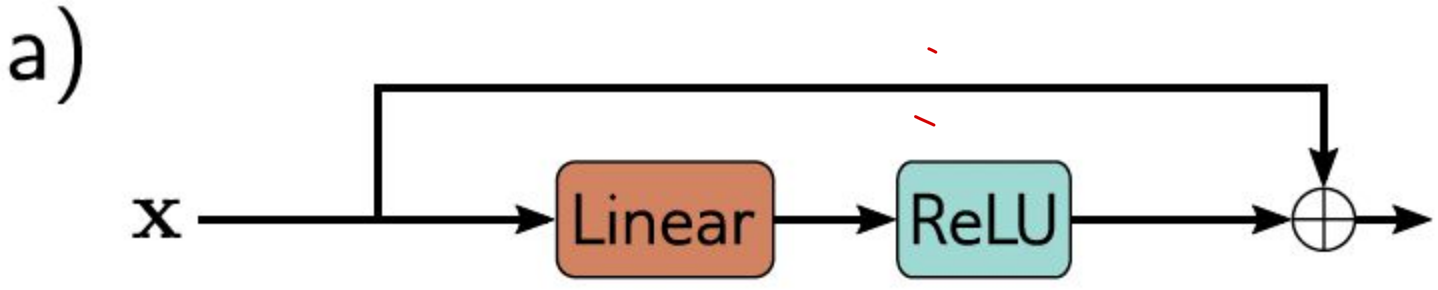
2^4

Upon expanding (unraveling) the network equations, we find that the output is the sum of the input plus four smaller networks (depicted in white, orange, gray, and cyan, respectively); we can think of this as an ensemble of networks.

Alternatively, we can consider the network as a combination of 16 different paths through the computational graph.

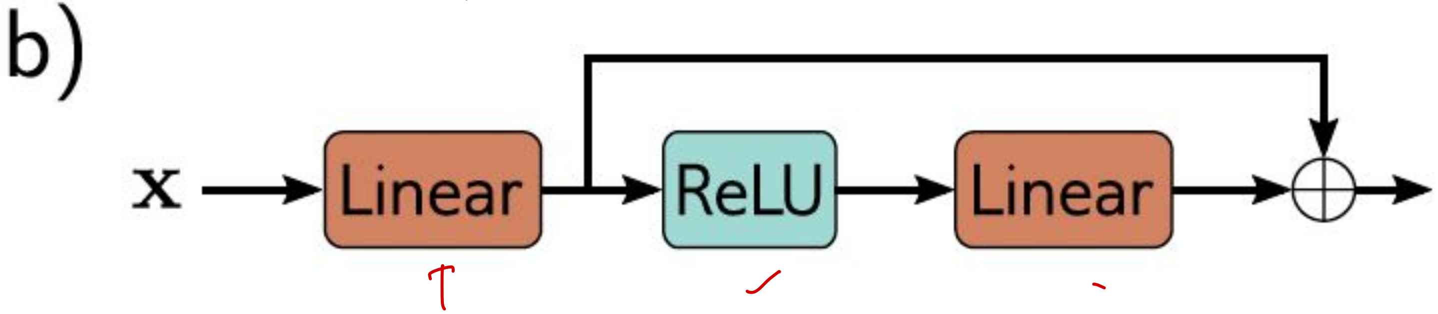
[Gradients through shorter paths behave better]

Order of operations in residual blocks



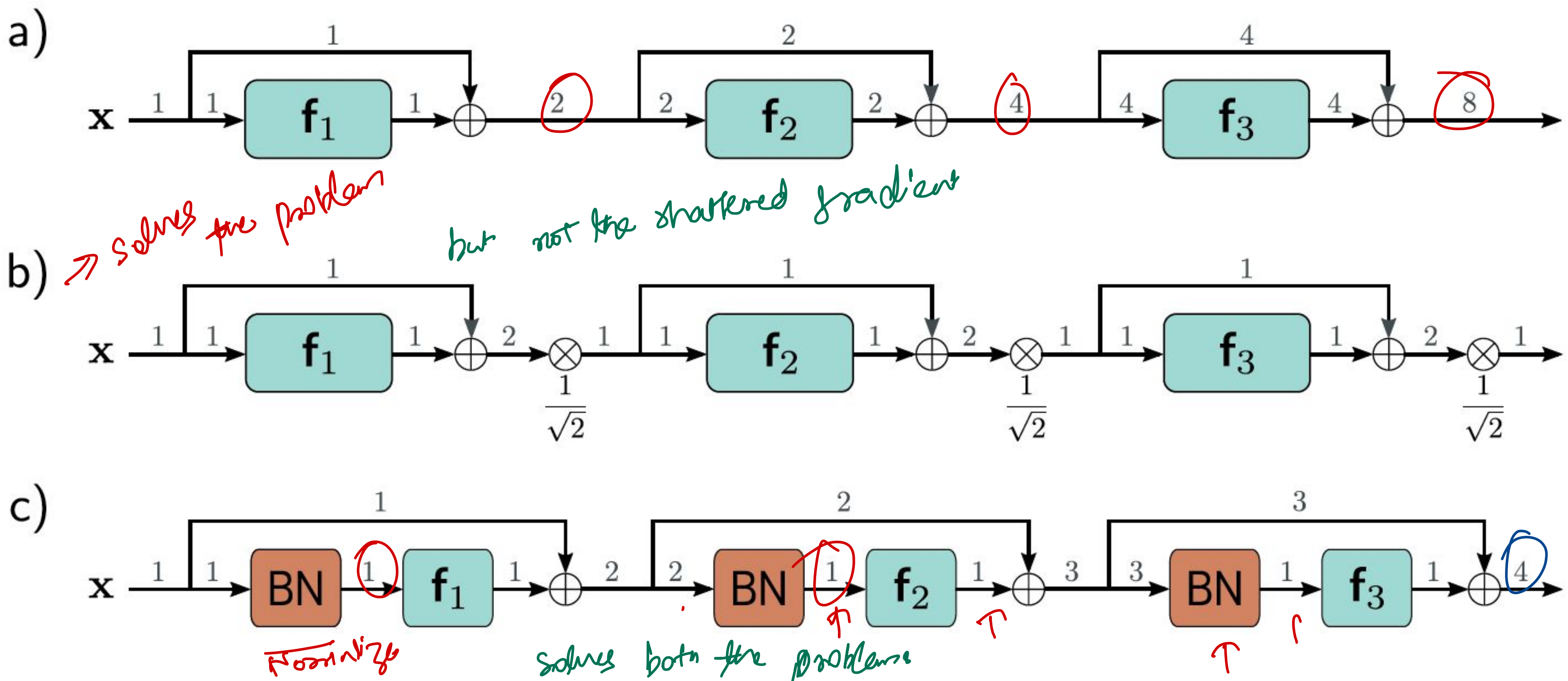
Linear transformation or convolution followed by a ReLU nonlinearity means that each residual block can only add non-negative quantities

Swap →



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

Exploding gradients



(a) The variance doubles at each layer (gray numbers indicate variance) and grows exponentially

Batch Normalization

Consider a single layer $y = Wx$

The following could lead to tough optimization:
- Inputs x are not *centered around zero* (need large bias)
- Inputs x have different scaling per-element (entries in W will need to vary a lot)

Idea: force inputs to be “nicely scaled” at each layer!

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want zero-mean unit-variance activations? just make them so."

consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

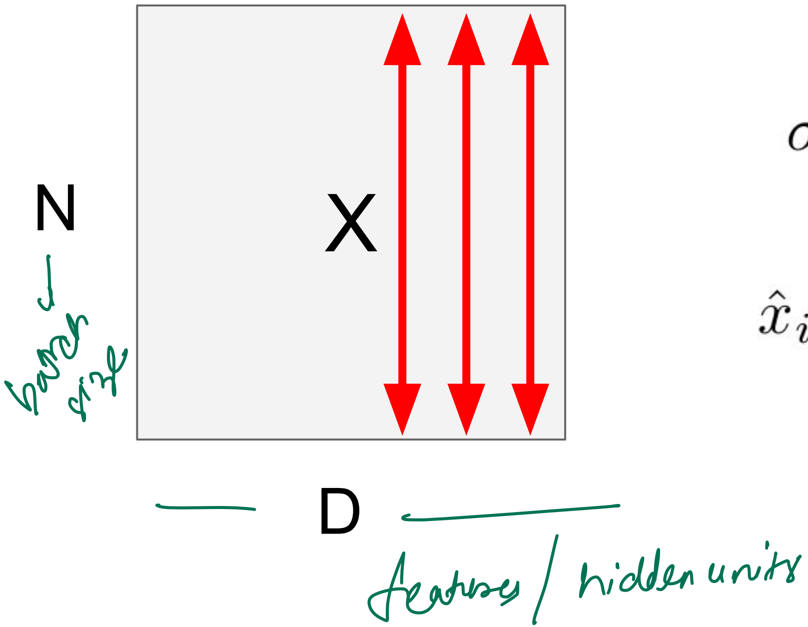
Batch Normalization

[Ioffe and Szegedy, 2015]

samples

size of hidden layers

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

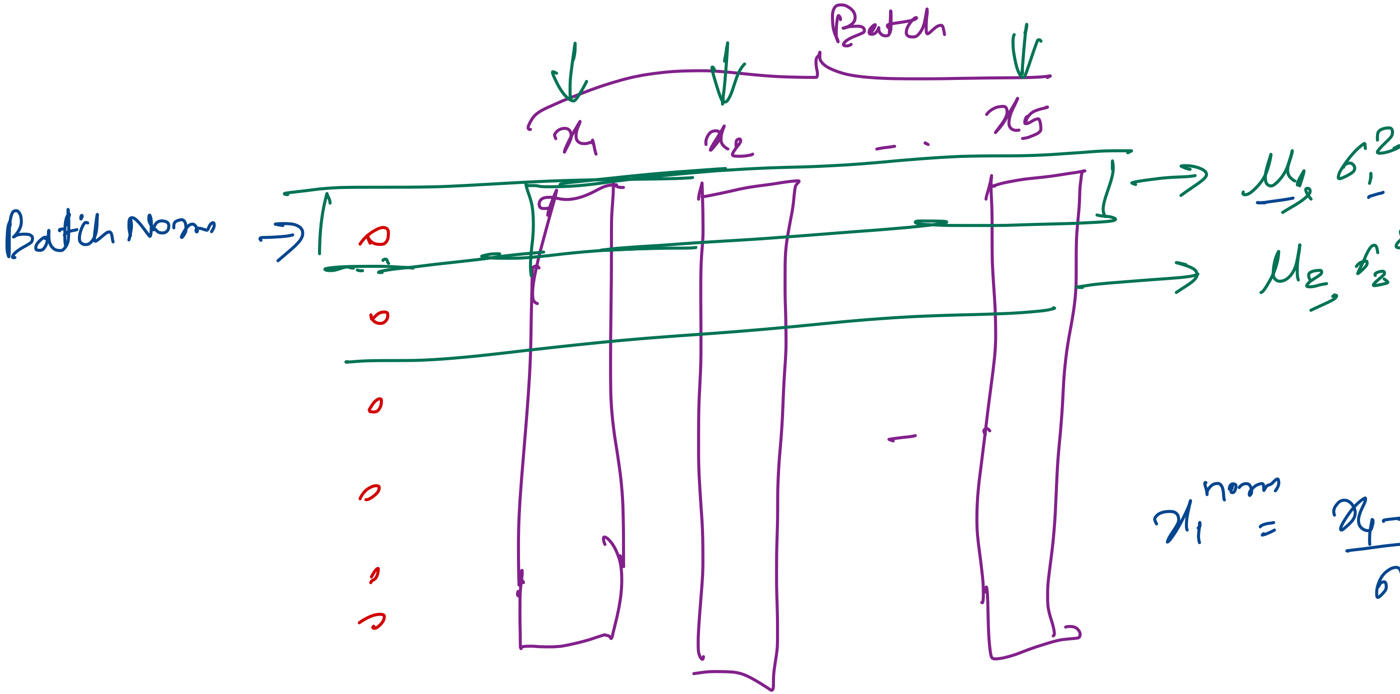
Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D



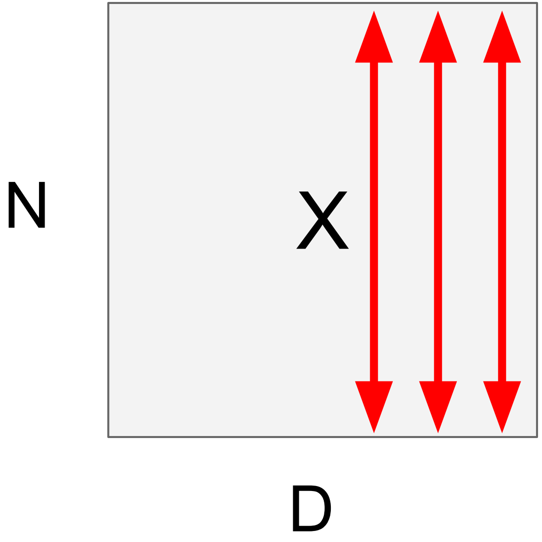
$$x_1^{norm} = \frac{x_1 - \mu_1}{\sigma_1}$$

6 hidden units

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

Learning $\boxed{\gamma = \sigma}$, $\boxed{\beta = \mu}$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

mean 0, var 1

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

var. γ_j^2 mean - β_j

Batch Normalization: Test-Time

Estimates depend on minibatch; can't do this at test-time!

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

During testing batchnorm becomes a linear operator! Can be fused with the previous fully-connected or conv layer

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

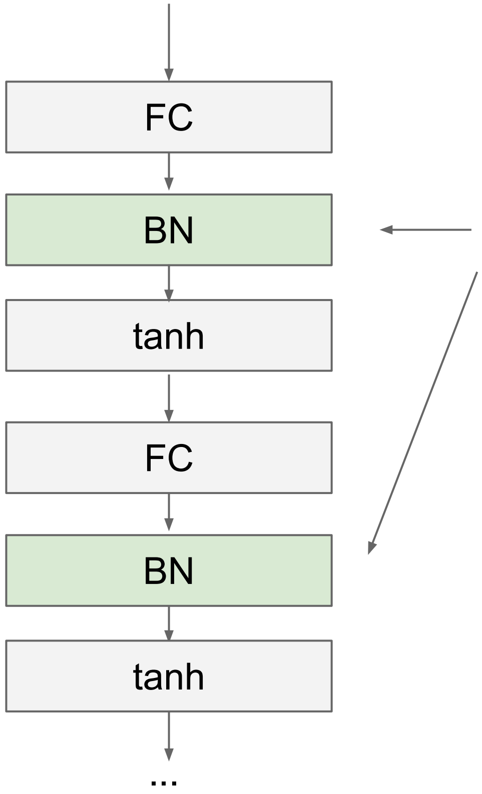
Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization

[Ioffe and Szegedy, 2015]

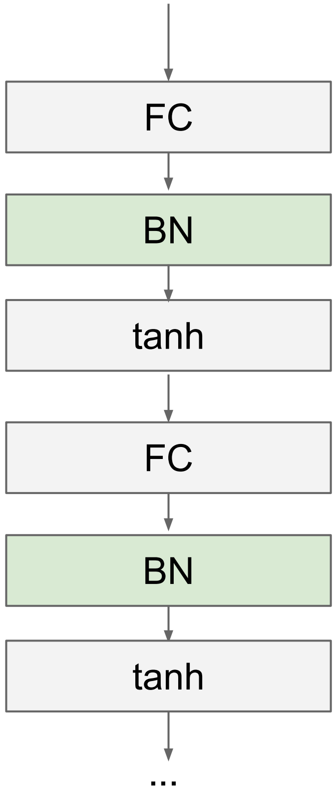


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization

[Ioffe and Szegedy, 2015]



- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$\mathbf{x}$: N × D

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}$: 1 × D

$\boldsymbol{\gamma}, \boldsymbol{\beta}$: 1 × D

$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\boldsymbol{\beta}$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$\mathbf{x}$: N×C×H×W

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}$: 1×C×1×1

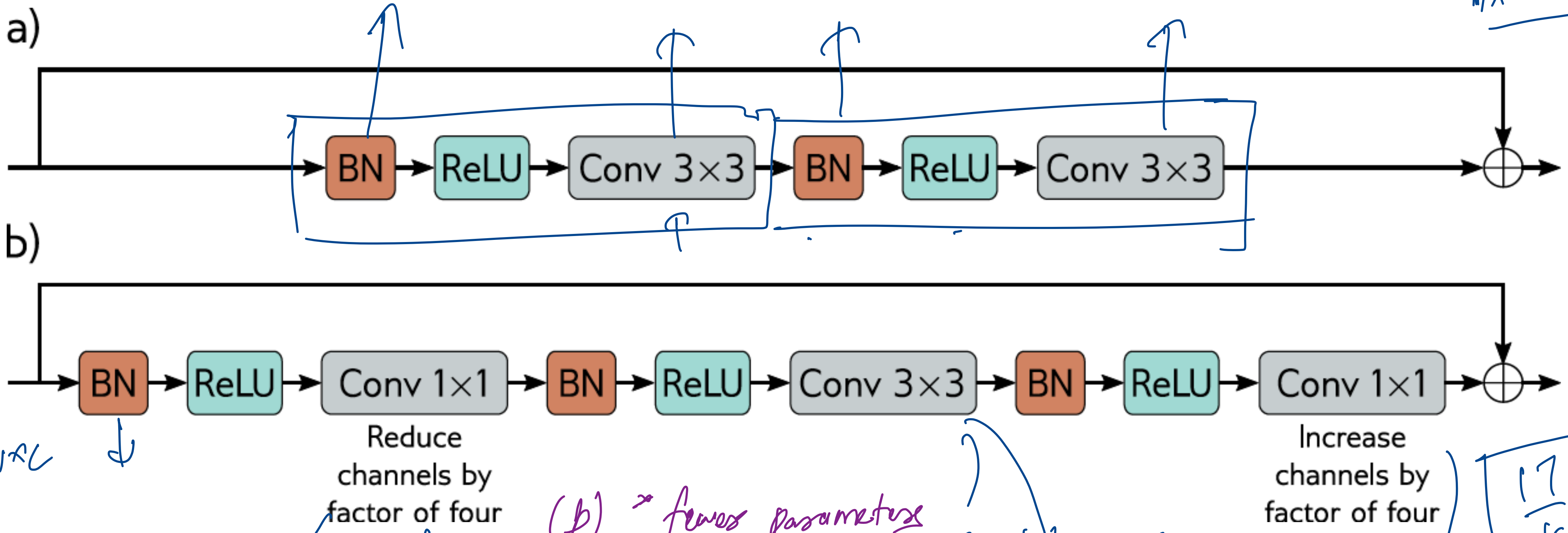
$\boldsymbol{\gamma}, \boldsymbol{\beta}$: 1×C×1×1

$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\boldsymbol{\beta}$

Resnet Block

$18C^2 + 4C$

$H \times W \times C$ $2 * C$ $3 \times 3 \times C \times C$ $2 * C$ $3 \times 3 \times C \times C$ $H \times W \times C$



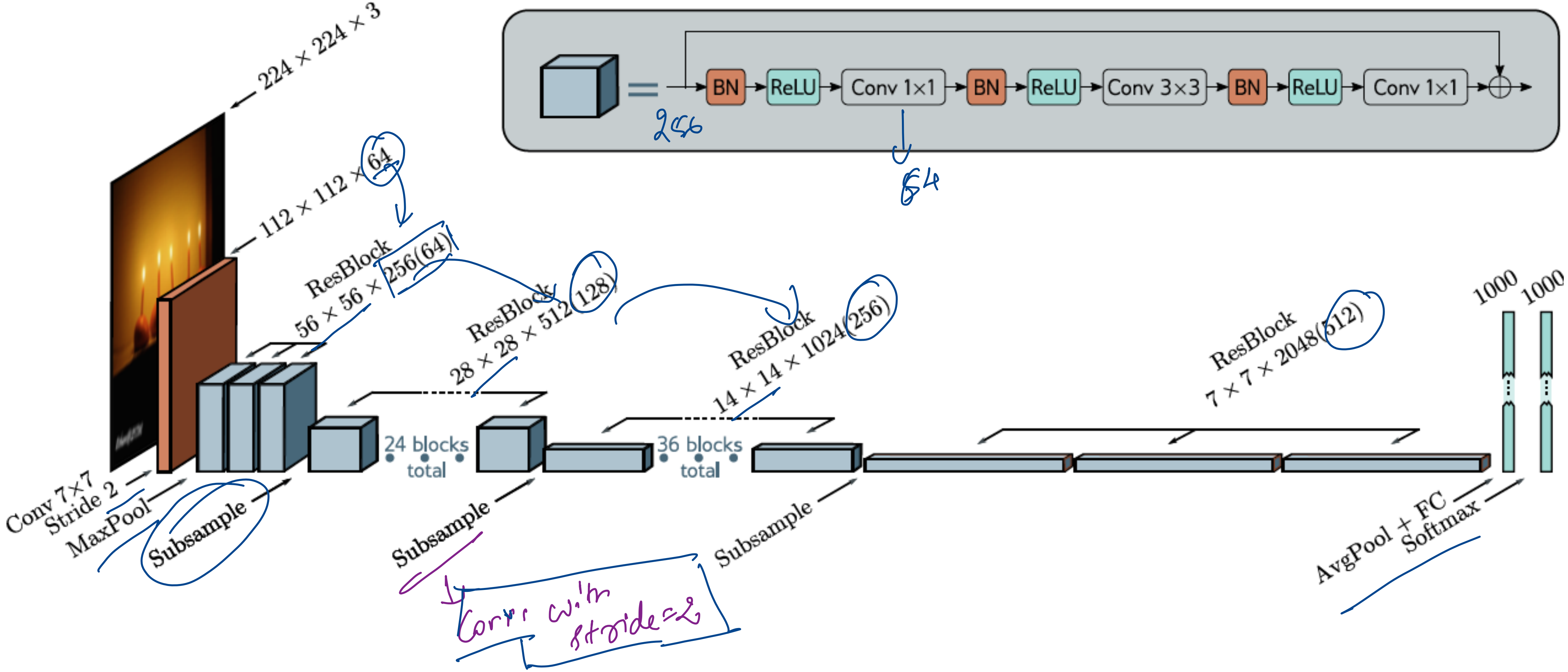
$H \times W \times C$

(b) * fewer parameters

$\frac{C^2}{4}$ $\frac{C}{4}$ $\frac{C}{4}$ $9\frac{C^2}{16}$ $\frac{C^2}{4}$ C $\frac{17}{16}C^2$ $\approx C^2$

Resnet 200 (2016)

v2



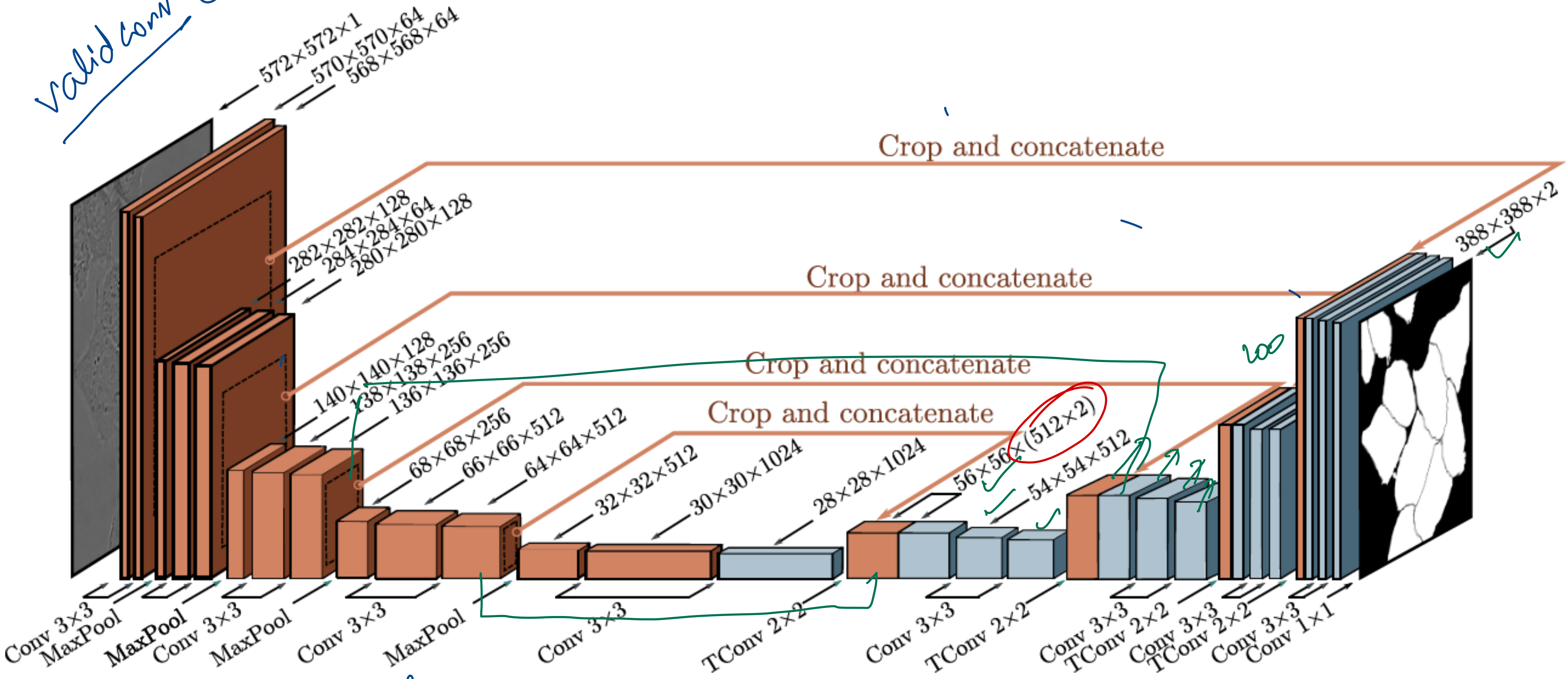
Convolution #2

- 2D Convolution
- Downsampling and upsampling, 1x1 convolution
- Image classification
- Object detection
- Semantic segmentation
- Residual networks
- U-Nets and hourglass networks

U-Net (2016)

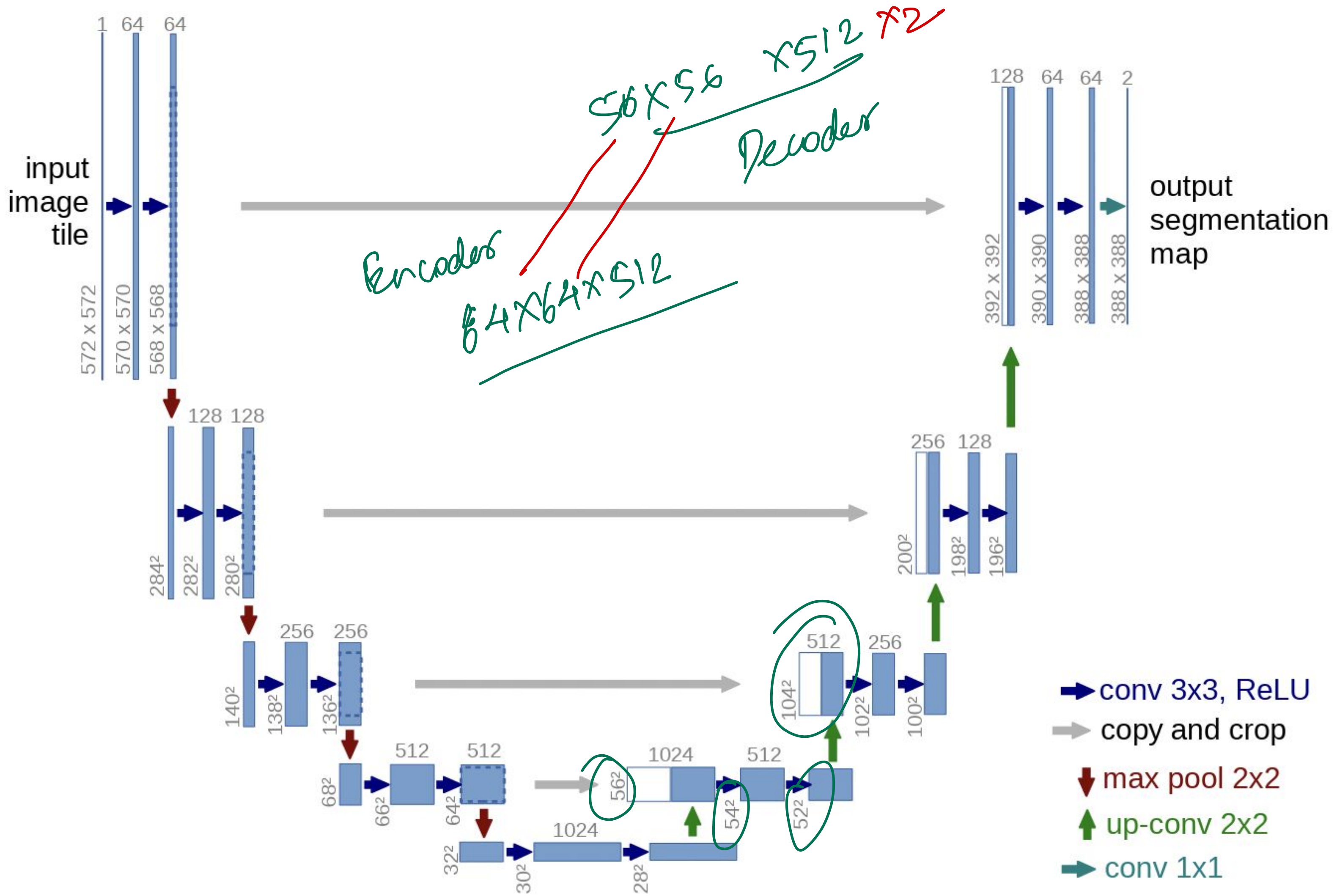
hourglass network

valid conv.

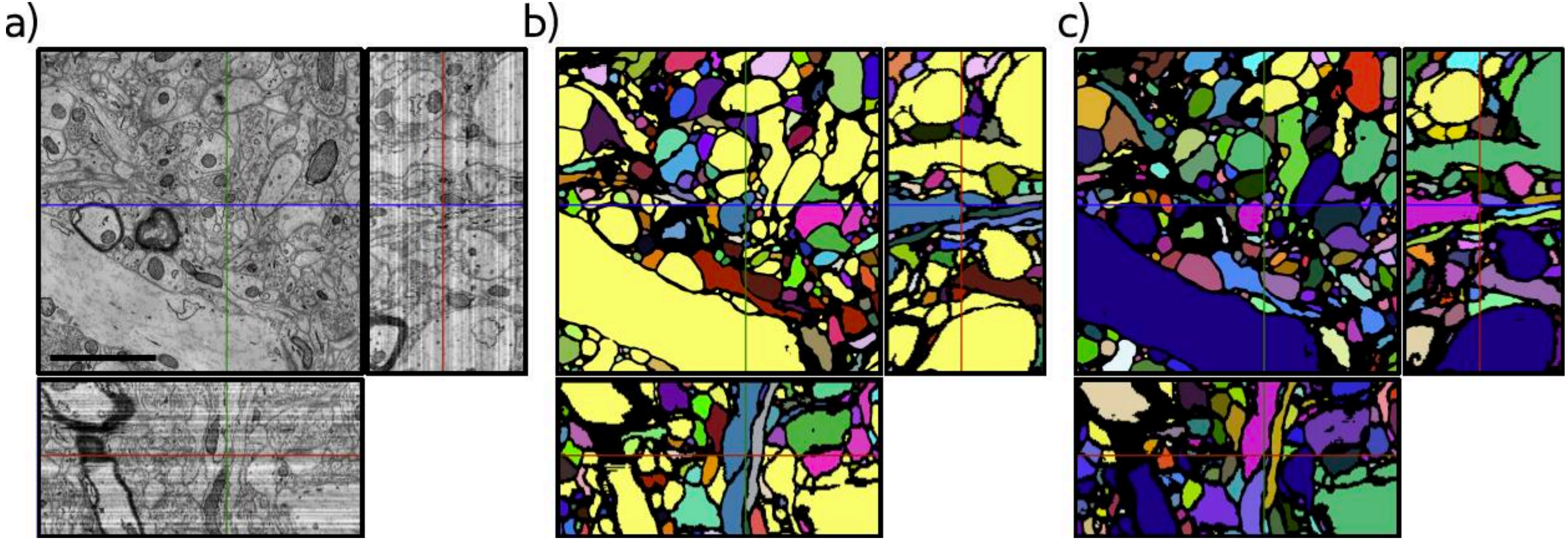


Encoder

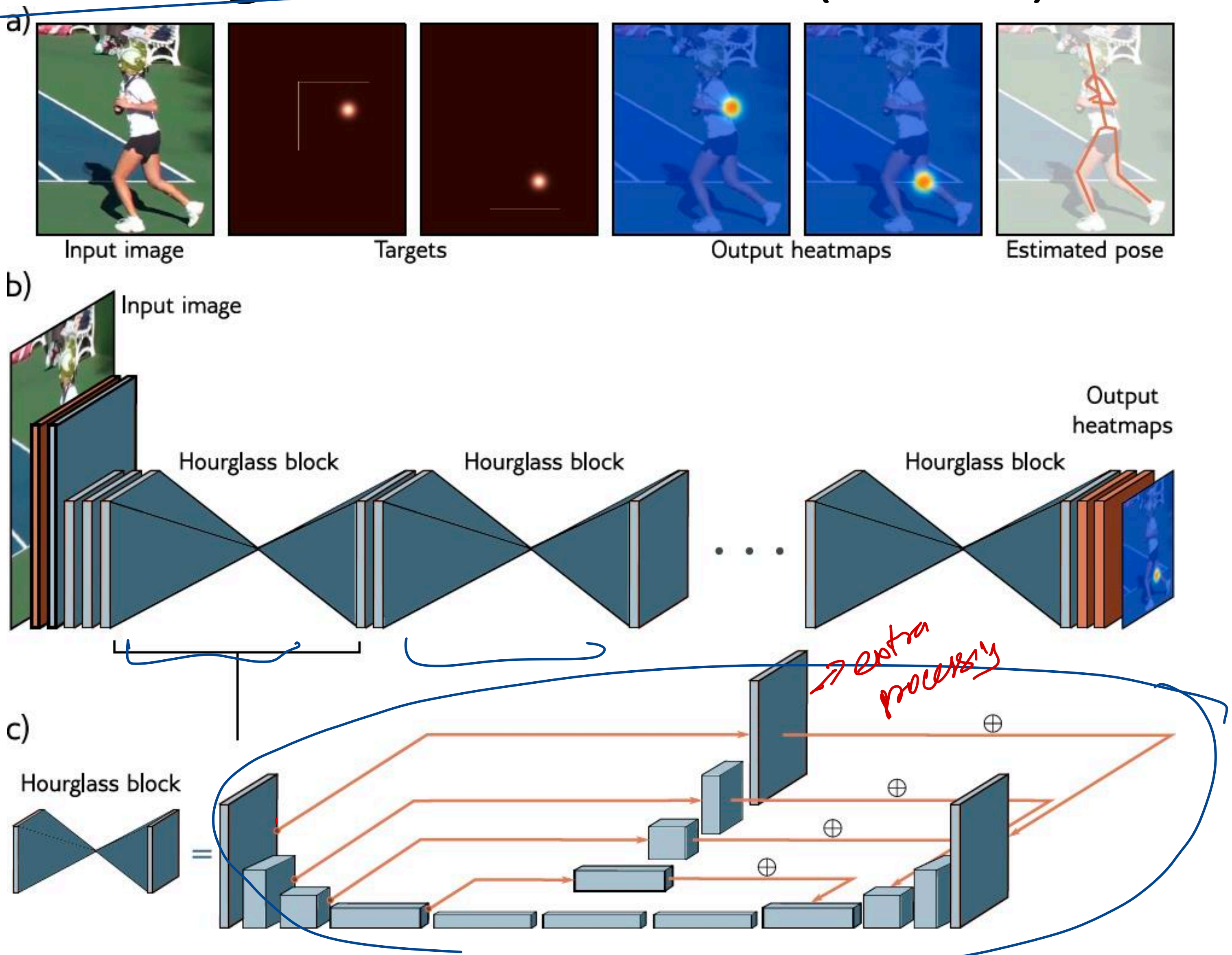
Decoder


input image tile
output segmentation map
1 64 64
572 x 572
570 x 570
568 x 568
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024 512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1
Encoder
Decoder
64x64x512
56x56 x512 x2

U-Net Results



Stacked hourglass networks (2016)



Crop and Concatenate

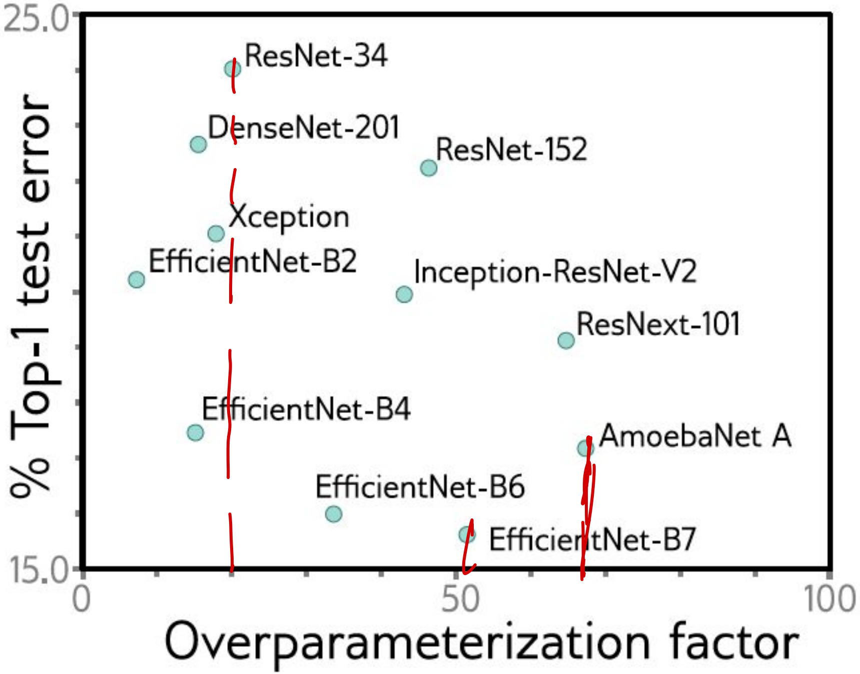
Basically, we take the difference in height and width between our target (which is the larger slice from the downwards path) and the goal (which is the smaller slice from the upwards path) and divide it equally, so the same amount of pixels is cut from the top, bottom, left and right.

```
def determine_crop(target, goal):
  height = (target.get_shape()[1] - goal.get_shape()[1]).value
  if height % 2 != 0:
    height_top, height_bottom = int(height/2), int(height/2) + 1
  else:
    height_top, height_bottom = int(height/2), int(height/2)

  width = (target.get_shape()[2] - goal.get_shape()[2]).value
  if width % 2 != 0:
    width_left, width_right = int(width/2), int(width/2) + 1
  else:
    width_left, width_right = int(width/2), int(width/2)



    return (height_top, height_bottom), (width_left, width_right)
```

The result of our `determine_crop` function can then be fed directly to a Cropping2D layer in Keras.

https://www.paepper.com/blog/posts/deep-learning-on-medical-images-with-u-net

Overparameterization



$= \frac{\#\text{ params}}{\#\text{ training data points}}$

Figure 20.3 Overparameterization. ImageNet performance for convolutional nets as a function of overparameterization (in multiples of dataset size). Most models have 10–100 times more parameters than there were training examples. Models compared are ResNet (He et al., 2016a,b), DenseNet (Huang et al., 2017b), Xception (Chollet, 2017), EfficientNet (Tan & Le, 2019), Inception (Szegedy et al., 2017), ResNeXt (Xie et al., 2017), and AmoebaNet (Cubuk et al., 2019).

Sejnowski (2020) suggests that ". . . the degeneracy of solutions changes the nature of the problem from finding a needle in a haystack to a haystack of needles."

GRADIENT DESCENT PROVABLY OPTIMIZES OVER-PARAMETERIZED NEURAL NETWORKS

Simon S. Du*
Machine Learning Department
Carnegie Mellon University
ssdu@cs.cmu.edu

Xiyu Zhai*
Department of EECS
Massachusetts Institute of Technology
xiyuzhai@mit.edu

Barnabás Poczós
Machine Learning Department
Carnegie Mellon University
bapozos@cs.cmu.edu

Aarti Singh
Machine Learning Department
Carnegie Mellon University
aartisingh@cmu.edu

Randomly initialized SGD converges to a global minimum for shallow fully connected ReLU networks with a least squares loss with enough hidden units.

One of the mysteries in the success of neural networks is randomly initialized first order methods like gradient descent can achieve zero training loss even though the objective function is non-convex and non-smooth.

This paper demystifies this surprising phenomenon for two-layer fully connected ReLU activated neural networks. For an m hidden node shallow neural network with ReLU activation and n training data, we show as long as m is large enough and no two inputs are parallel, randomly initialized gradient descent converges to a globally optimal solution at a linear convergence rate for the quadratic loss function.

These theoretical results are intriguing but usually make unrealistic assumptions about the network structure. For example, Du et al. (2019a) show that residual networks converge to zero training loss when the width of the network D (i.e., the number of hidden units) is $\mathcal{O}[I^4 K^2]$ where I is the amount of training data, and K is the depth of the network. Similarly, Nguyen & Hein (2017) assume that the network's width is larger than the dataset size, which is unrealistic in most practical scenarios. Overparameterization seems to be important, but theory cannot yet explain empirical fitting performance.