CSC413/2516 Lecture 5: CNN Architectures, Batch/Layer Normalization, Residual Networks

Bo Wang

Overview

Convolutional Neural Networks

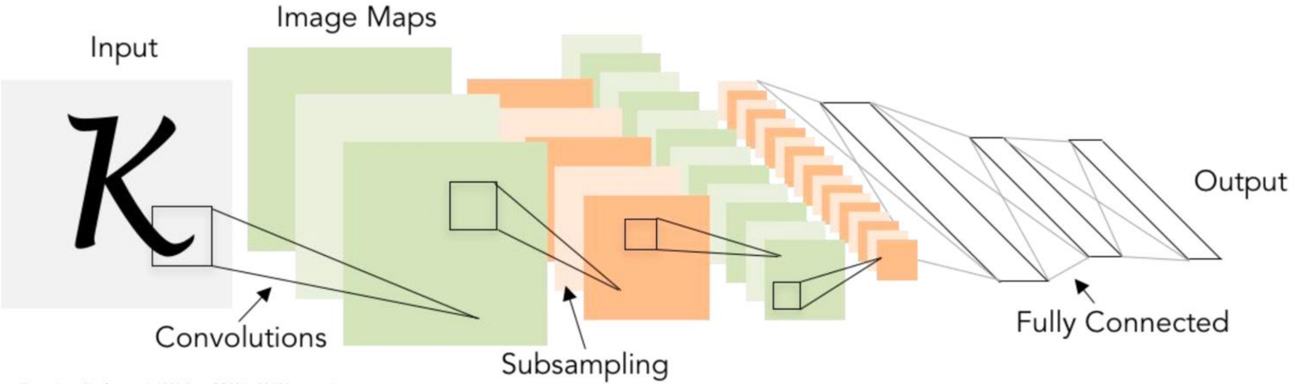
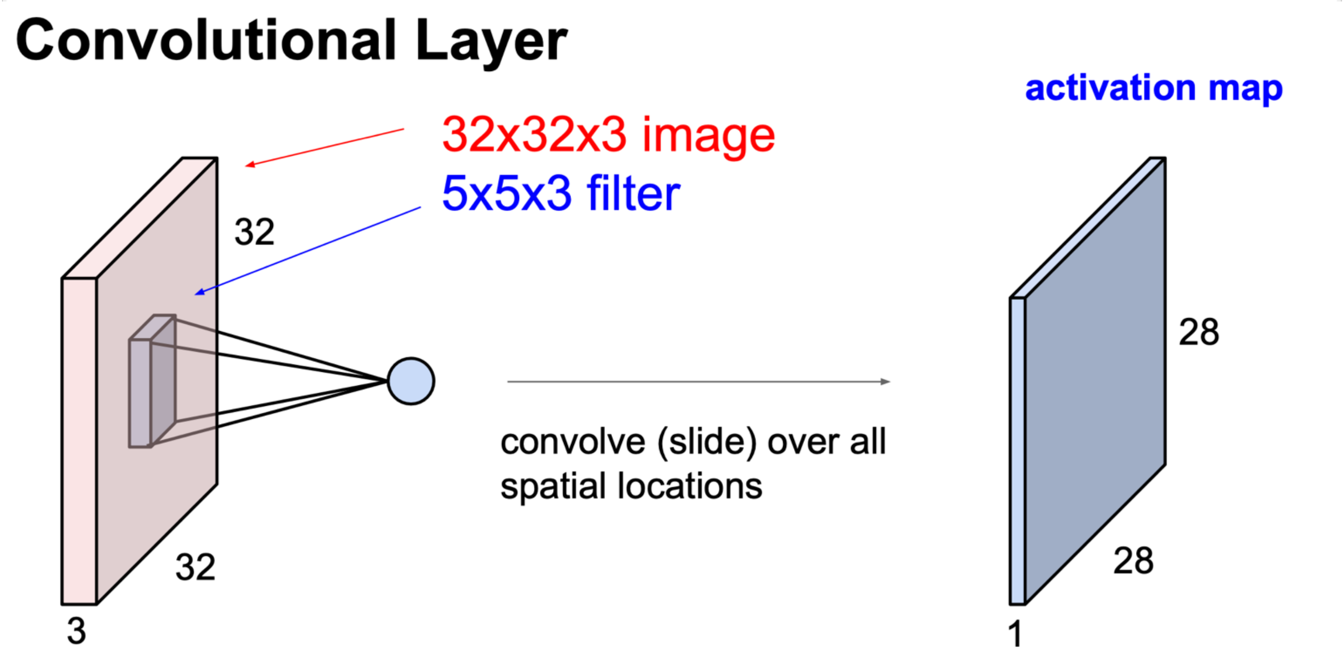


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Overview



Convolutional Layer

32x32x3 image

5x5x3 filter

convolve (slide) over all spatial locations

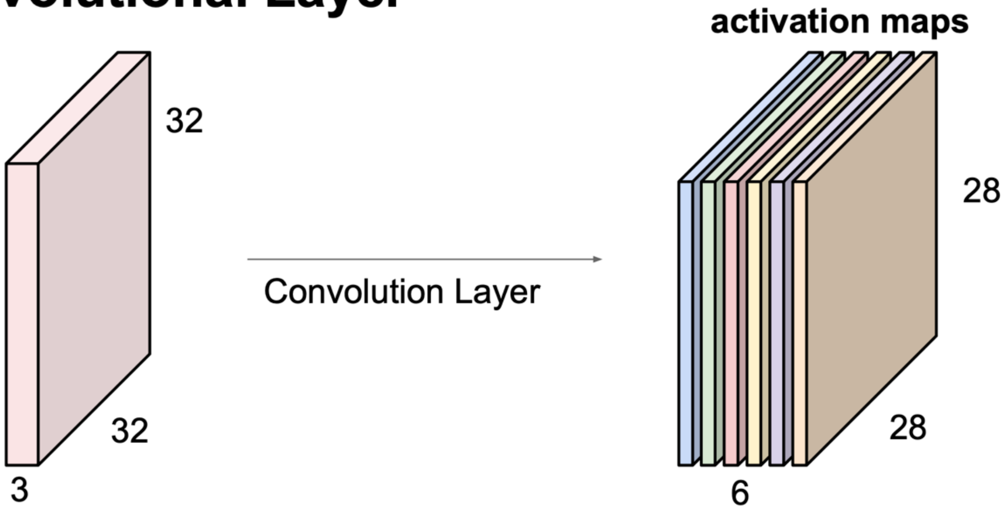
activation map

Source: https://cs231n.stanford.edu/

Overview

Convolutional Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

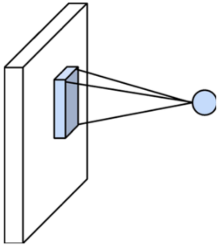


We stack these up to get a "new image" of size 28x28x6!

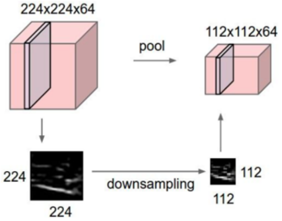
Overview

Components of Convolutional Neural Networks (CNNs)

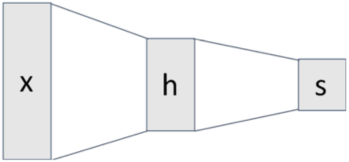
Convolution Layers



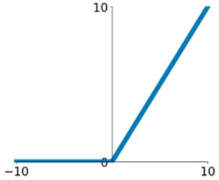
Pooling Layers



Fully-Connected Layers

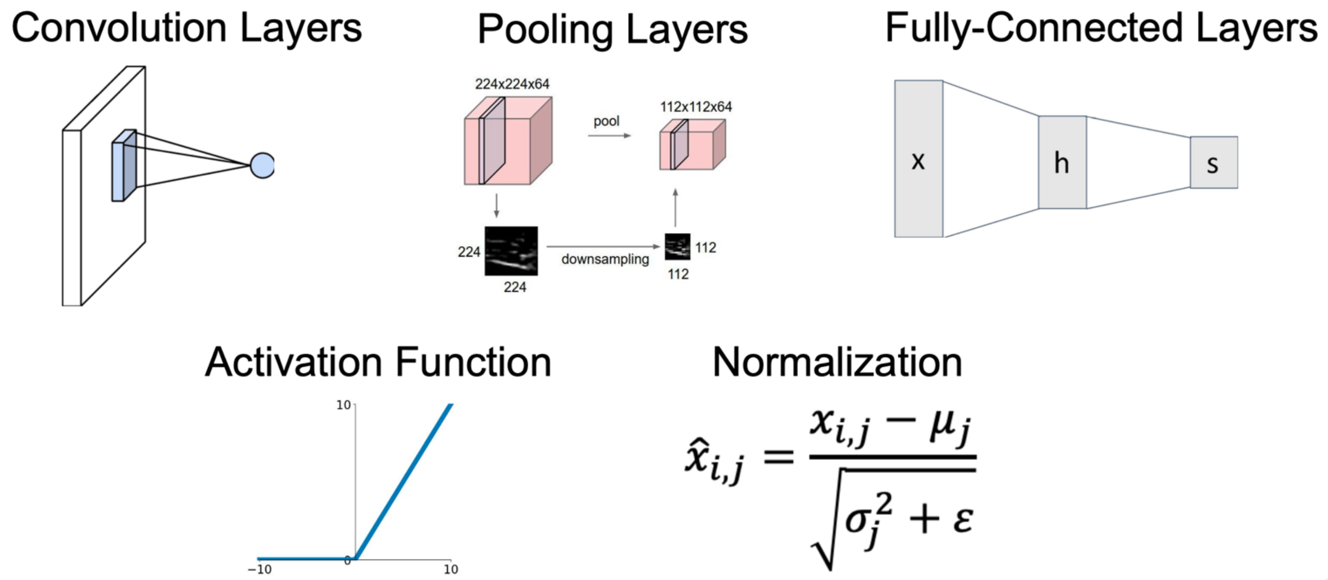


Activation Function



Overview

Components of Convolutional Neural Networks (CNNs)



Convolution Layers

Pooling Layers

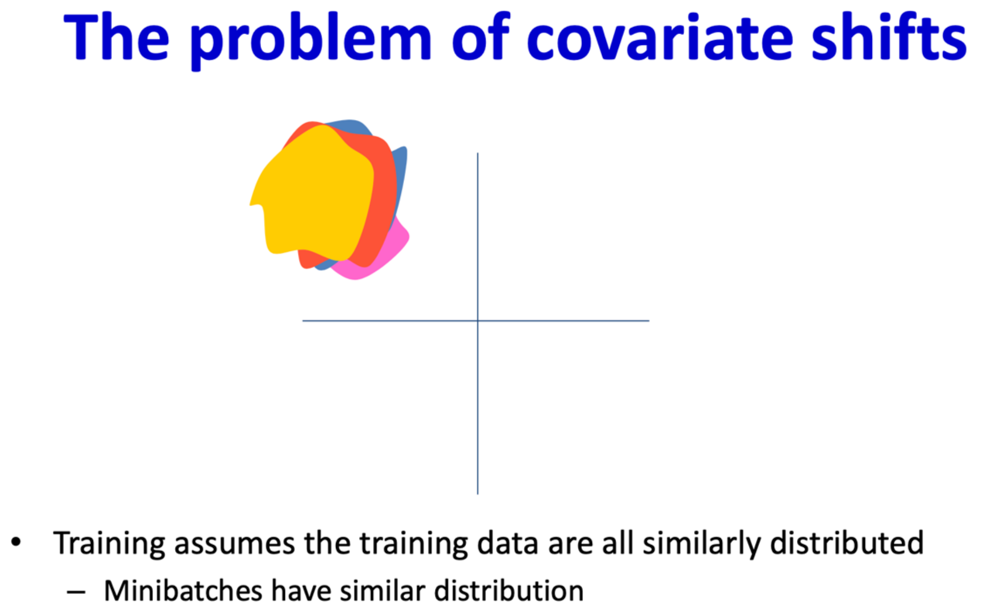
Fully-Connected Layers

Activation Function

Normalization

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

Batch Normalization

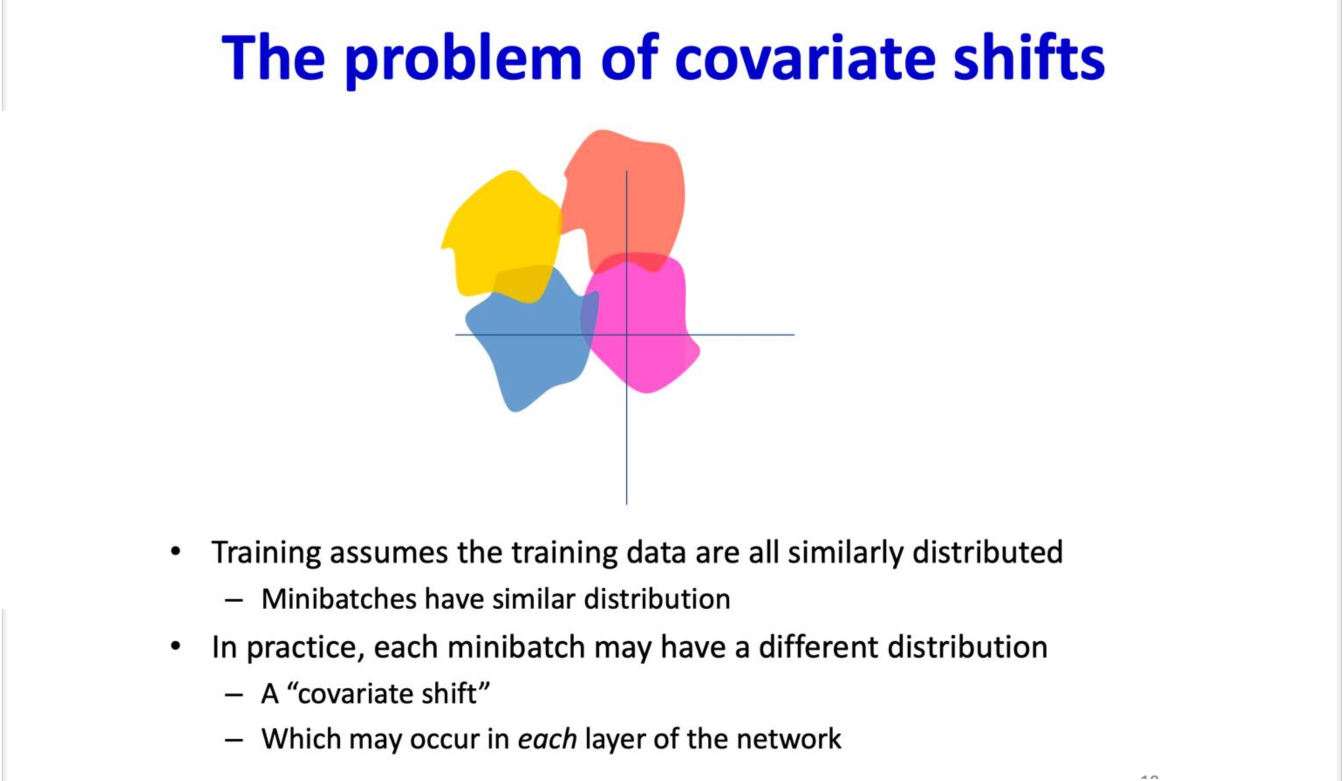


The problem of covariate shifts

- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution

Source: https://deeplearning.cs.cmu.edu/F20/

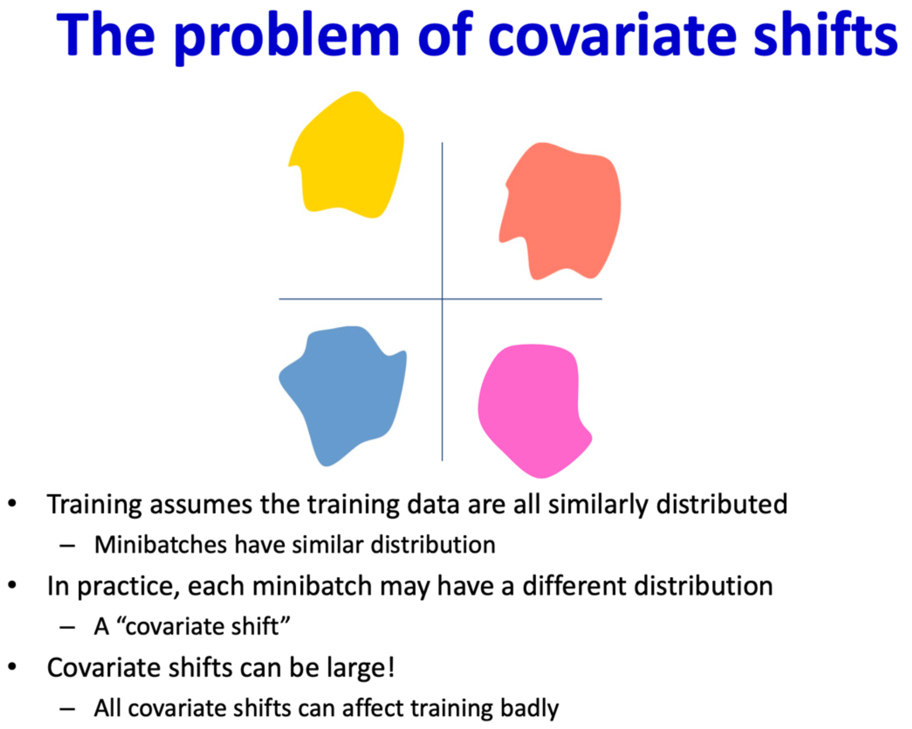
Batch Normalization



The problem of covariate shifts

- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution
- In practice, each minibatch may have a different distribution
  - A “covariate shift”
  - Which may occur in *each* layer of the network

Batch Normalization

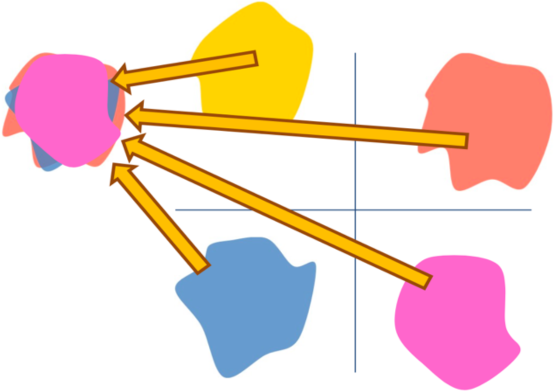


The problem of covariate shifts

- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution
- In practice, each minibatch may have a different distribution
  - A "covariate shift"
- Covariate shifts can be large!
  - All covariate shifts can affect training badly

Batch Normalization

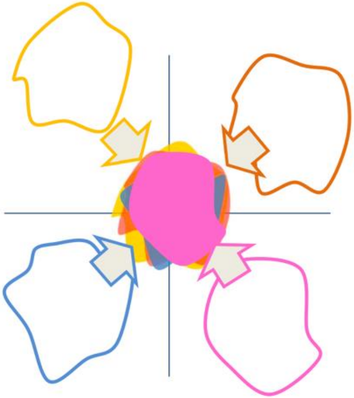
Solution: Move all minibatches to a "standard" location



- "Move" all batches to a "standard" location of the space
  - But where?
  - To determine, we will follow a two-step process

Batch Normalization

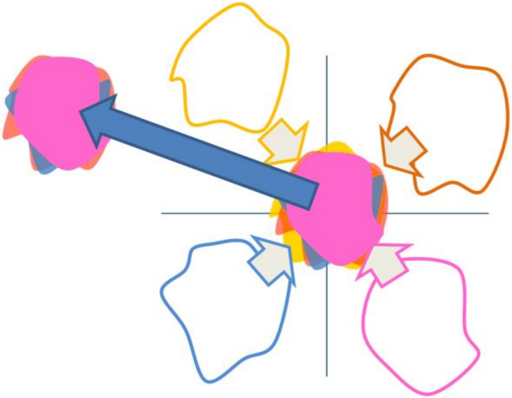
Move all minibatches to a "standard" location



- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

Batch Normalization

(Mini)Batch Normalization



- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches
- **Then move the entire collection to the appropriate location**

Batch Normalization

“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:

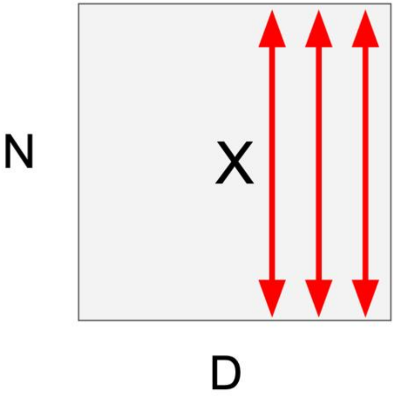
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

Source: https://cs231n.stanford.edu/

Batch Normalization

Input: $x : N \times D$



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

Per-channel mean, shape is D

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

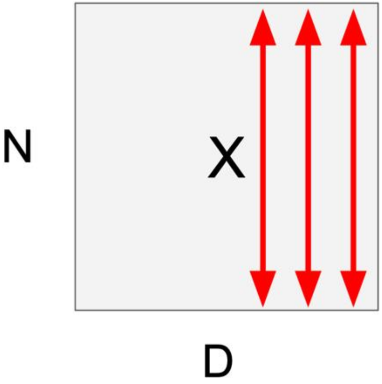
Per-channel var, shape is D

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

Normalized x, Shape is N x D

Batch Normalization

Input: $x : N \times D$



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

Per-channel mean, shape is D

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

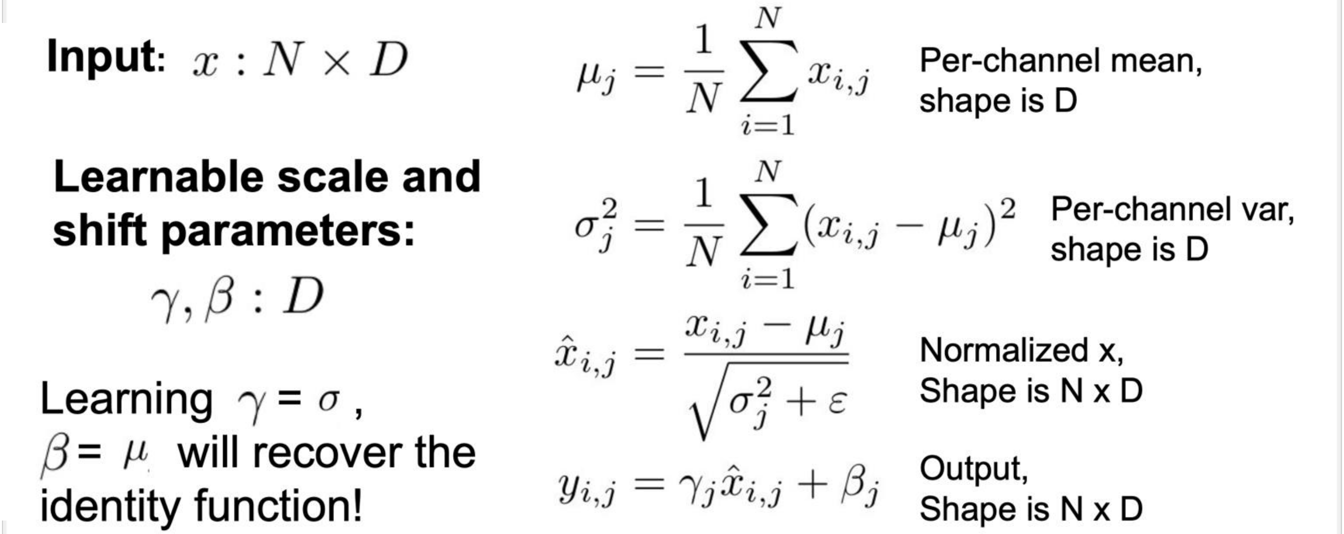
Per-channel var, shape is D

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

Normalized x, Shape is N x D

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

Batch Normalization



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

Batch Normalization: Test-Time

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

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

Learnable scale and shift parameters:

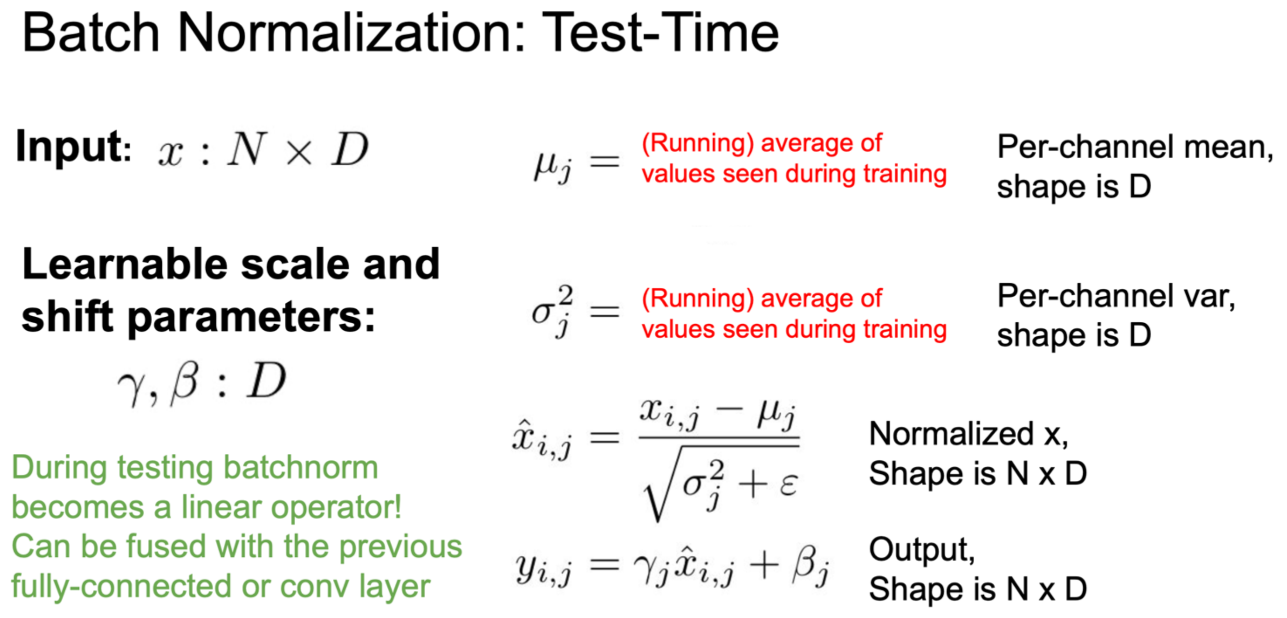
$$\gamma, \beta : D$$

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

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

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

Batch Normalization



Batch Normalization: Test-Time

Input: $x : N \times D$

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

Learnable scale and shift parameters:

$\gamma, \beta : D$

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

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

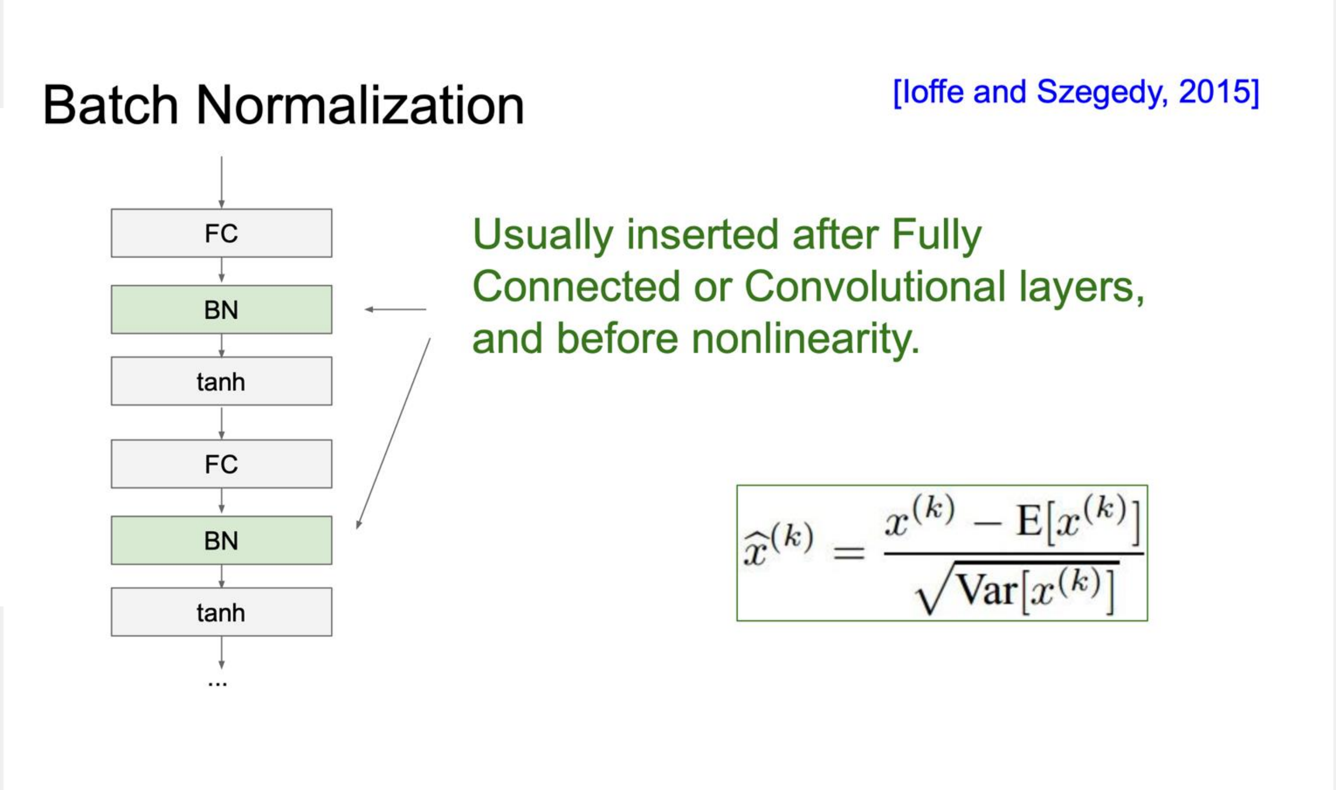
Normalized x, Shape is N x D

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

Output, Shape is N x D

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

Batch Normalization



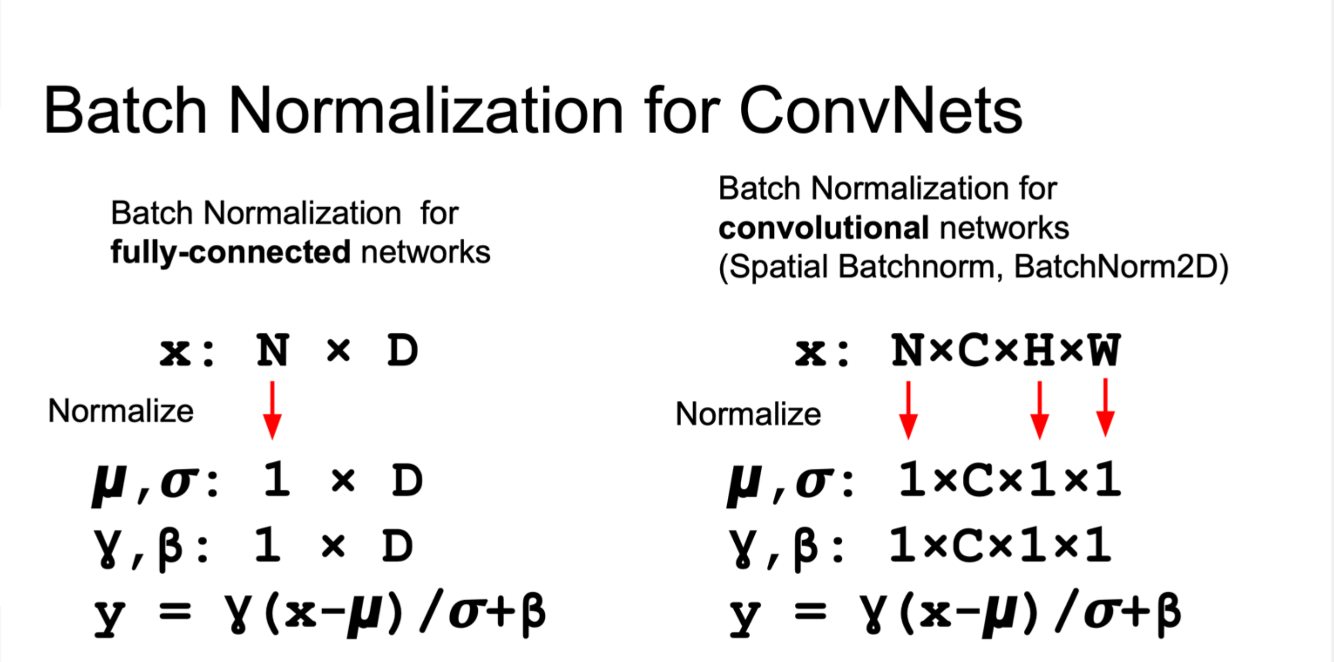
Batch Normalization

[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



Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$x: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

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

$x: N \times C \times H \times W$

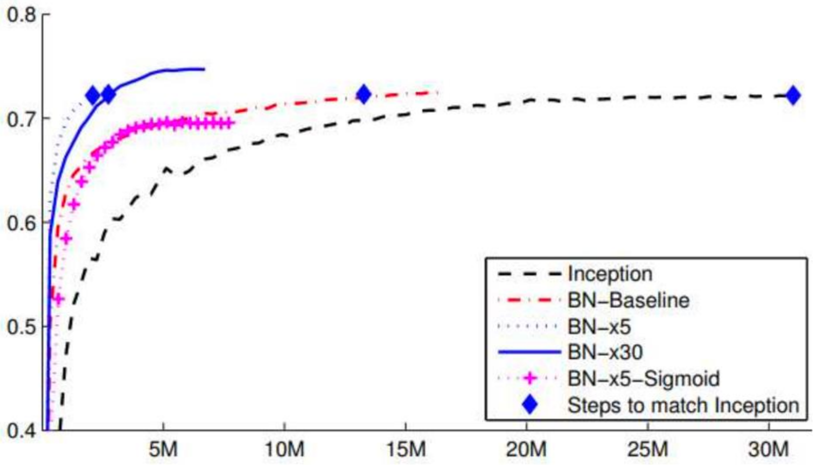
Normalize

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

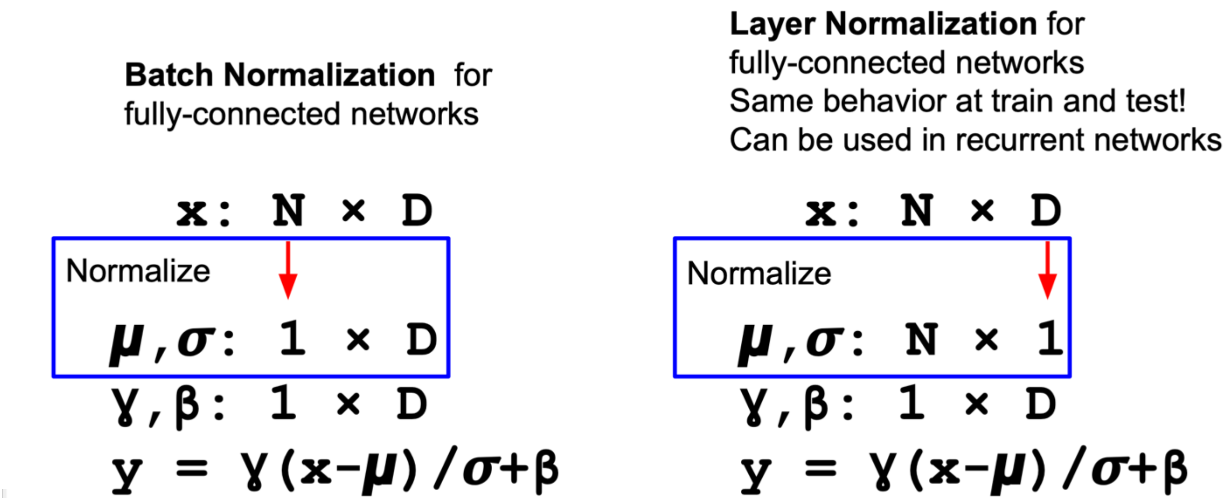
$y = \gamma(x-\mu)/\sigma+\beta$

Batch Normalization



- Performance on Imagenet, from Ioffe and Szegedy, JMLR 2015

Layer Normalization



Batch Normalization for fully-connected networks

$$\mathbf{x}: N \times D$$

Normalize

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

$$\gamma, \beta: 1 \times D$$

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

Layer Normalization for fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$$\mathbf{x}: N \times D$$

Normalize

$$\boldsymbol{\mu}, \boldsymbol{\sigma}: N \times 1$$

$$\gamma, \beta: 1 \times D$$

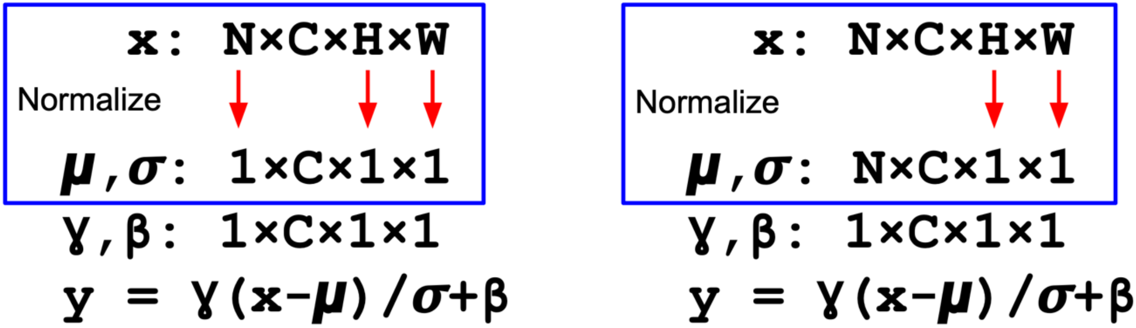
$$y = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$$

Ba, Kiros, and Hinton, “Layer Normalization”, arXiv 2016

Layer Normalization

Batch Normalization for convolutional networks

Layer Normalization for convolutional networks



$x: N \times C \times H \times W$

Normalize

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x - \mu)/\sigma + \beta$

$x: N \times C \times H \times W$

Normalize

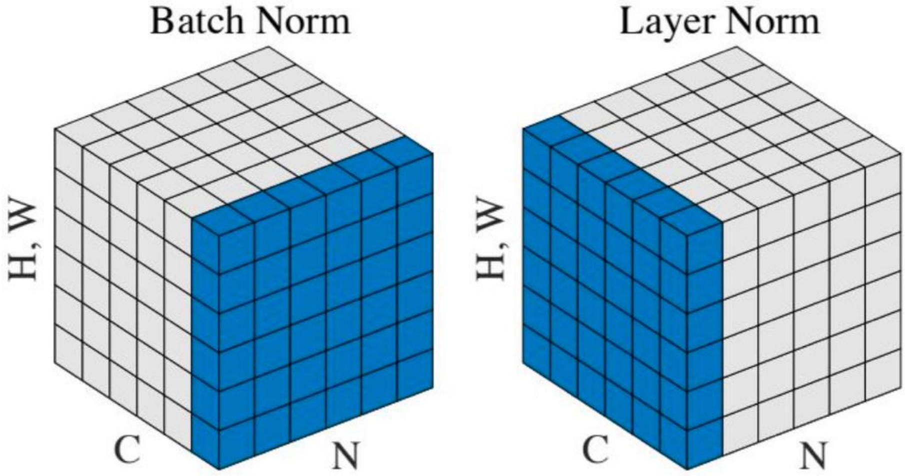
$\mu, \sigma: N \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

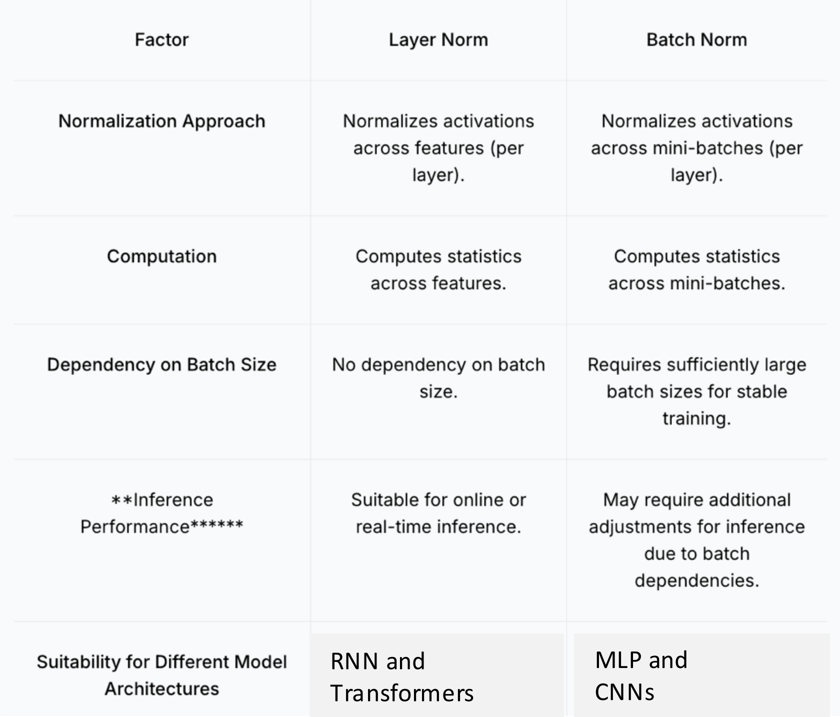
$y = \gamma(x - \mu)/\sigma + \beta$

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Batch vs Layer Normalization



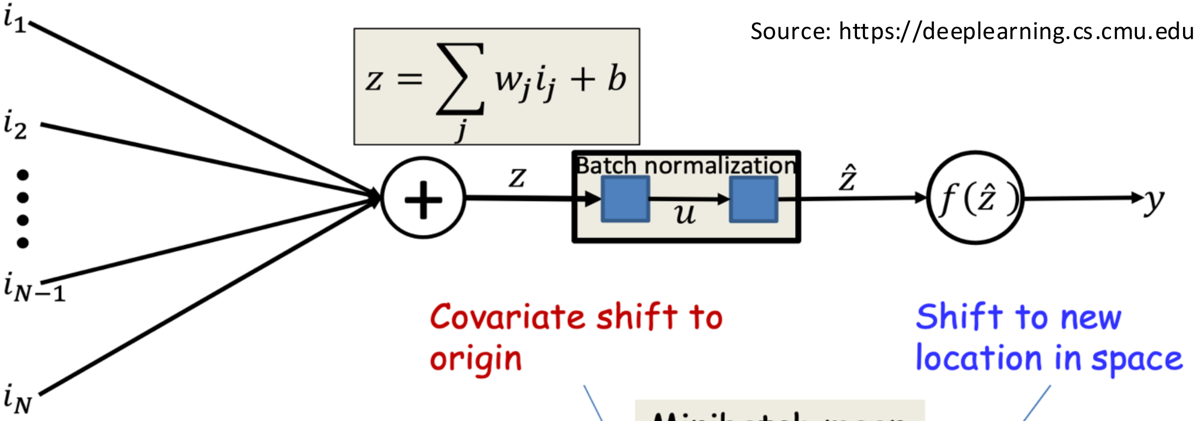
Batch vs Layer Normalization



Factor	Layer Norm	Batch Norm
Normalization Approach	Normalizes activations across features (per layer).	Normalizes activations across mini-batches (per layer).
Computation	Computes statistics across features.	Computes statistics across mini-batches.
Dependency on Batch Size	No dependency on batch size.	Requires sufficiently large batch sizes for stable training.
∗∗Inference Performance******	Suitable for online or real-time inference.	May require additional adjustments for inference due to batch dependencies.
Suitability for Different Model Architectures	RNN and Transformers	MLP and CNNs

After Break: Training with Batch Normalization

Batch Normalization -- Backpropagation



Source: https://deeplearning.cs.cmu.edu/F20/

$$z = \sum_j w_j i_j + b$$

Covariate shift to origin

Shift to new location in space

Minibatch mean

$$u_i = \frac{z_i - \mu_B}{\sigma_B}$$

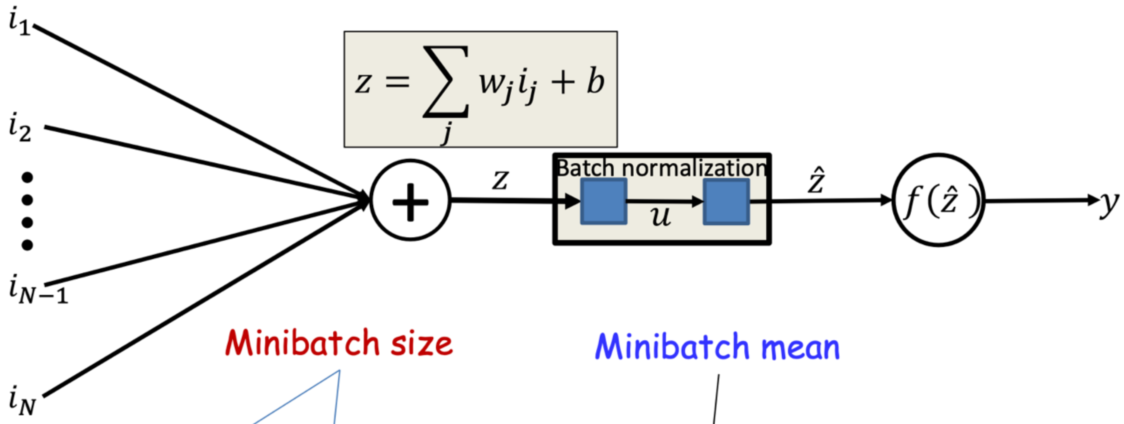
$$\hat{z}_i = \gamma u_i + \beta$$

Minibatch standard deviatiation

Neuron-specific terms

- BN aggregates the statistics over a minibatch and normalizes the batch by them
- Normalized instances are "shifted" to a *unit-specific* location

Batch Normalization -- Backpropagation



$$z = \sum_j w_j i_j + b$$

Minibatch size

Minibatch mean

Minibatch standard deviation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

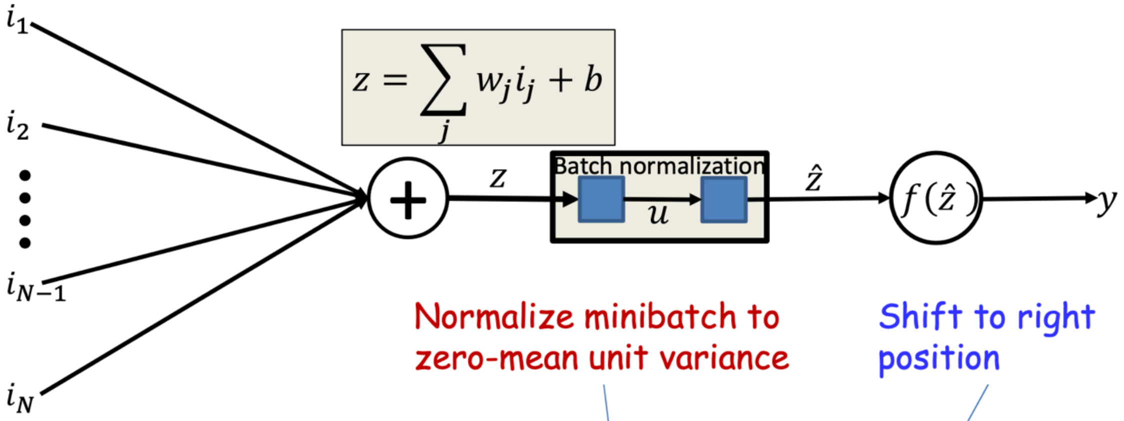
$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B} (z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

- BN aggregates the statistics over a minibatch and normalizes the batch by them
- Normalized instances are “shifted” to a *unit-specific* location

Batch Normalization -- Backpropagation



Normalize minibatch to zero-mean unit variance

Shift to right position

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

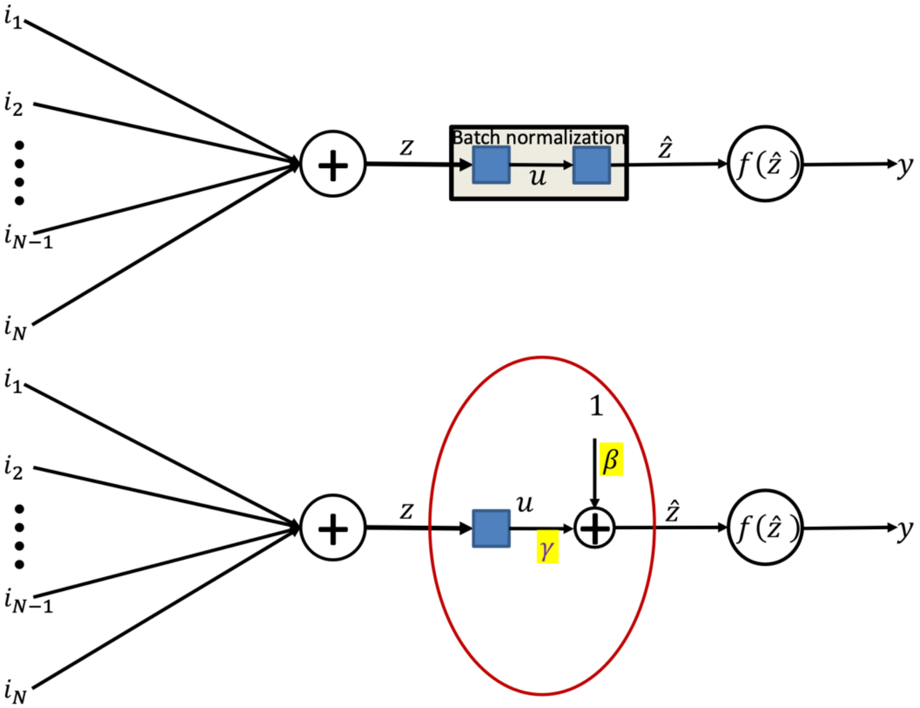
$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

- BN aggregates the statistics over a minibatch and normalizes the batch by them
- Normalized instances are “shifted” to a *unit-specific* location

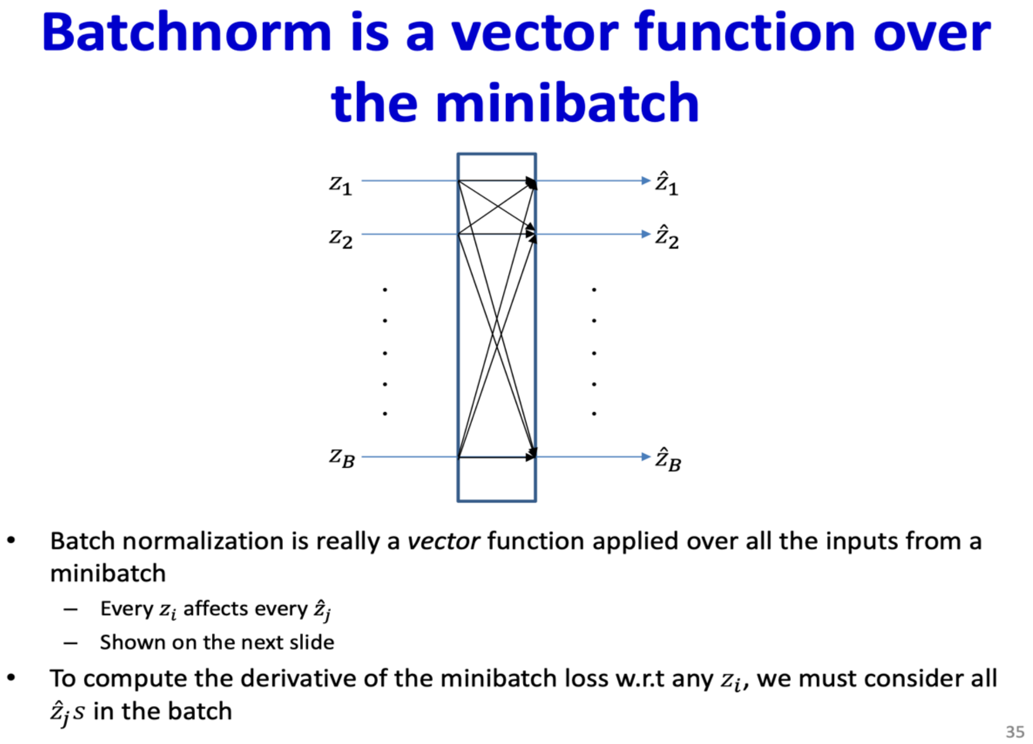
Batch Normalization -- Backpropagation

A better picture for batch norm



31

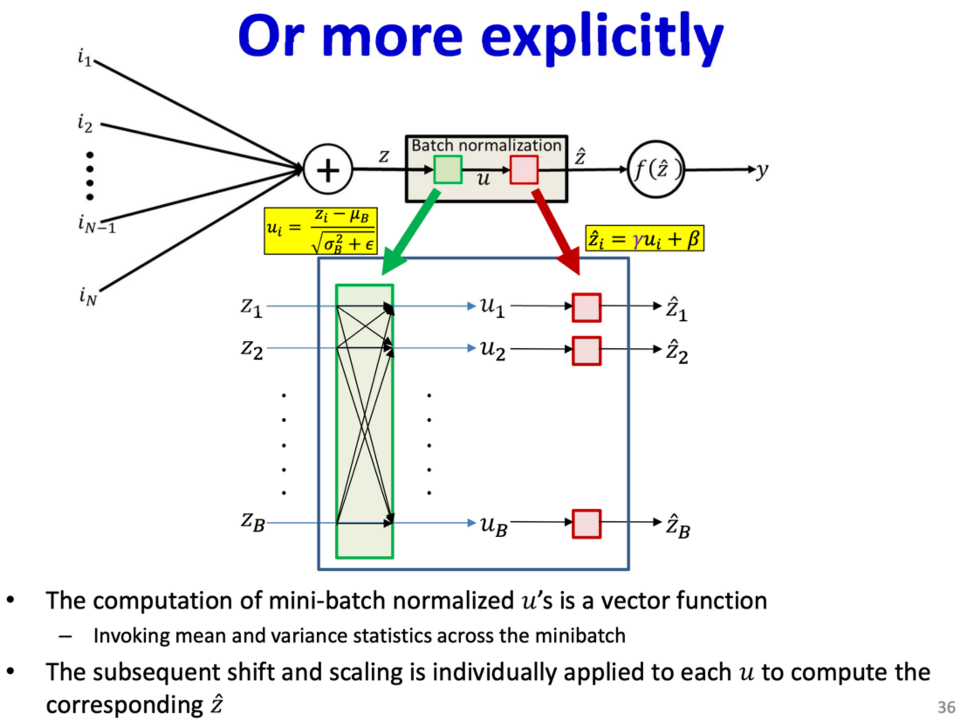
Batch Normalization -- Backpropagation



Batchnorm is a vector function over the minibatch

- Batch normalization is really a *vector* function applied over all the inputs from a minibatch
  - Every z_i affects every $\hat{z}_j$
  - Shown on the next slide
- To compute the derivative of the minibatch loss w.r.t any z_i, we must consider all $\hat{z}_j s$ in the batch

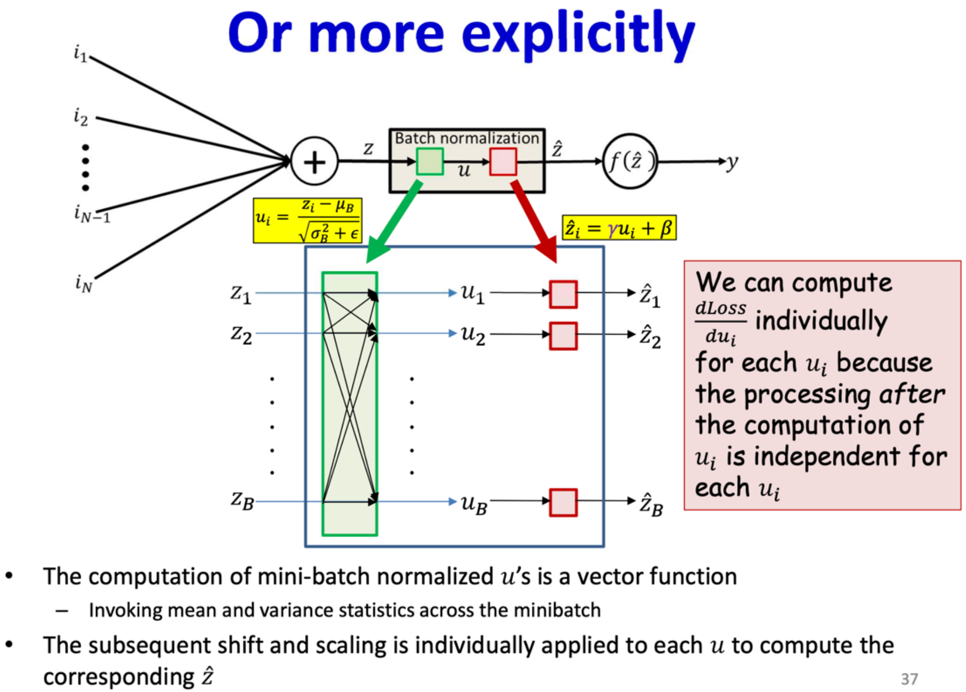
Batch Normalization -- Backpropagation



Or more explicitly

- The computation of mini-batch normalized u's is a vector function
  - Invoking mean and variance statistics across the minibatch
- The subsequent shift and scaling is individually applied to each u to compute the corresponding $\hat{z}$

Batch Normalization -- Backpropagation



Or more explicitly

$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

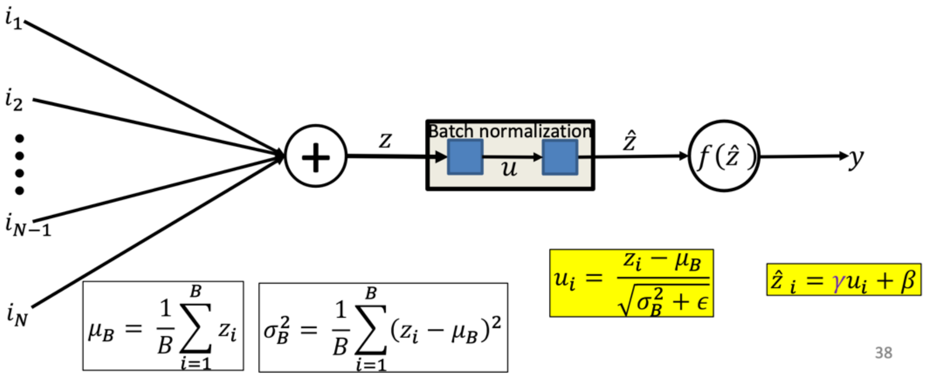
$\hat{z}_i = \gamma u_i + \beta$

We can compute $\frac{dLoss}{du_i}$ individually for each u_i because the processing *after* the computation of u_i is independent for each u_i

- The computation of mini-batch normalized u's is a vector function
  - Invoking mean and variance statistics across the minibatch
- The subsequent shift and scaling is individually applied to each u to compute the corresponding $\hat{z}$

37

Batch Normalization -- Backpropagation



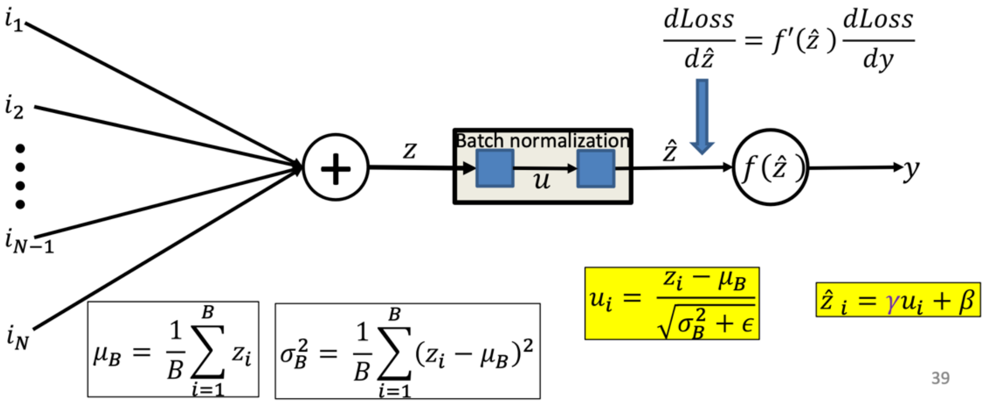
$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

Batch Normalization -- Backpropagation



$$\frac{dLoss}{d\hat{z}} = f'(\hat{z})\frac{dLoss}{dy}$$

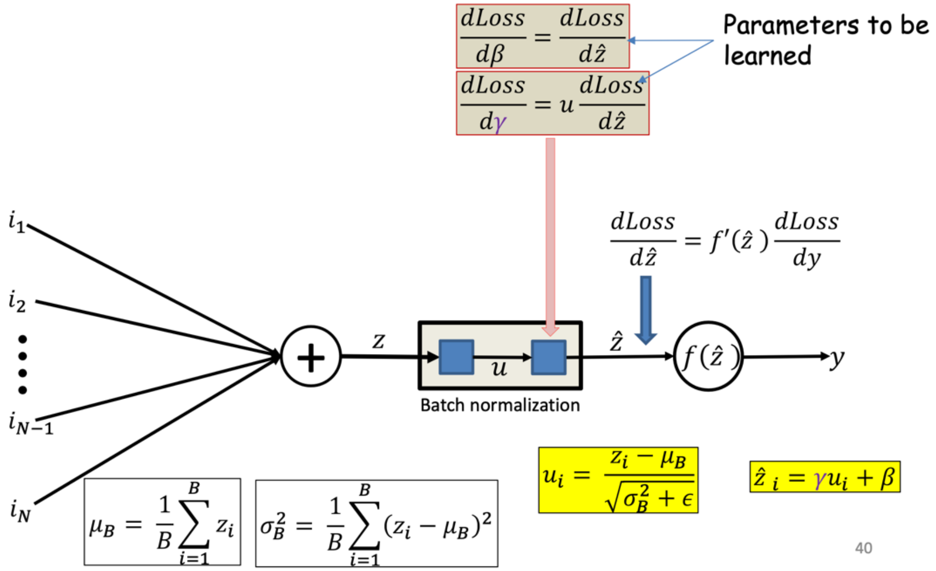
$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

Batch Normalization -- Backpropagation



$$\frac{dLoss}{d\beta} = \frac{dLoss}{d\hat{z}}$$

$$\frac{dLoss}{d\gamma} = u\frac{dLoss}{d\hat{z}}$$

Parameters to be learned

$$\frac{dLoss}{d\hat{z}} = f'(\hat{z})\frac{dLoss}{dy}$$

Batch normalization

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

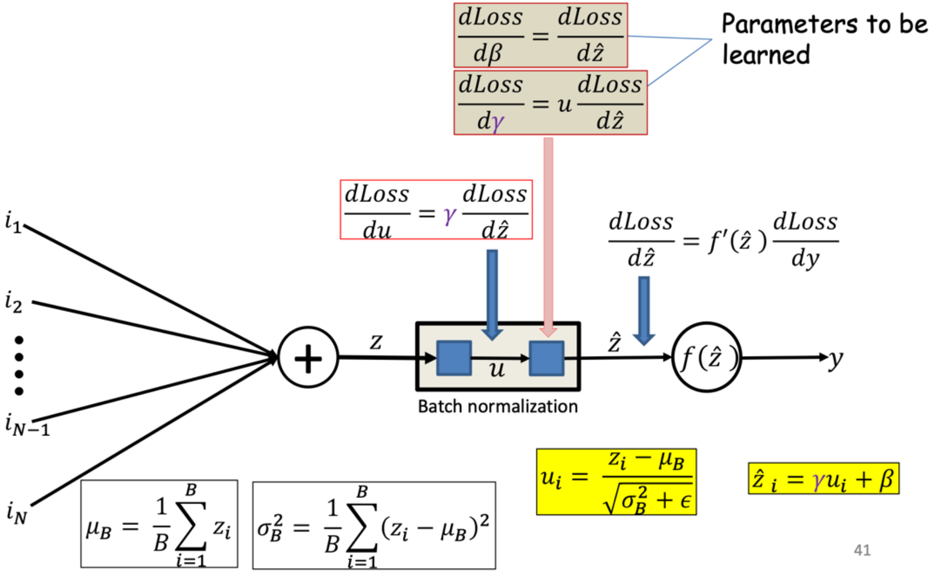
$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

40

Batch Normalization -- Backpropagation



$$\frac{dLoss}{d\beta} = \frac{dLoss}{d\hat{z}}$$

$$\frac{dLoss}{d\gamma} = u\frac{dLoss}{d\hat{z}}$$

Parameters to be learned

$$\frac{dLoss}{du} = \gamma\frac{dLoss}{d\hat{z}}$$

$$\frac{dLoss}{d\hat{z}} = f'(\hat{z})\frac{dLoss}{dy}$$

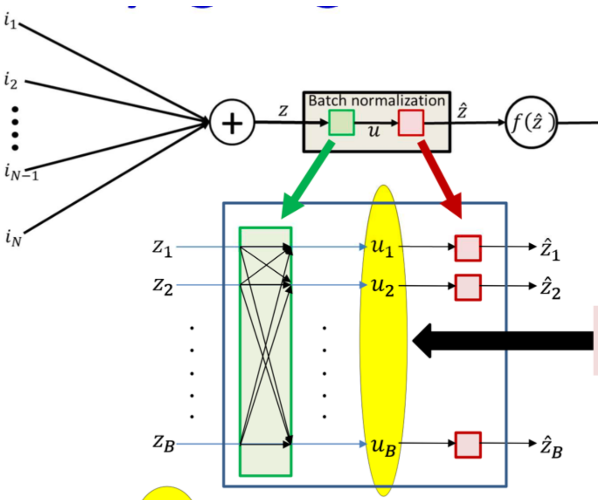
$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

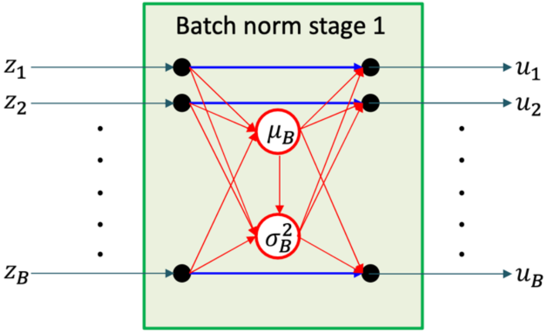
Batch Normalization -- Backpropagation



Derivatives computed for every u

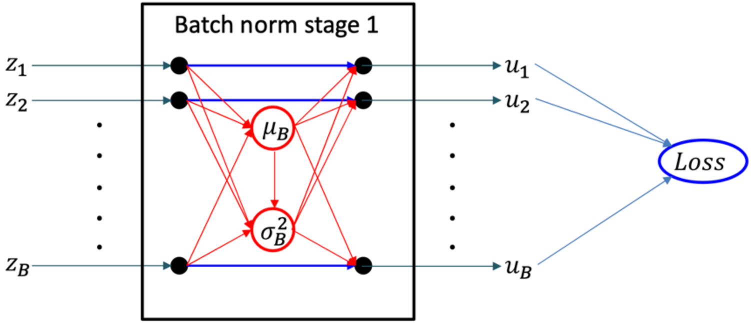
- We now have $\frac{dLoss}{du_i}$ for every u_i
- We must propagate the derivative through the first stage of BN
  - Which is a vector operation over the minibatch

Batch Normalization -- Backpropagation



- The complete dependency figure for the first “normalization” stage of Batchnorm
  - Which computes the centered “u”s from the “z”s for the minibatch
- Note : inputs and outputs are different *instances* in a minibatch
  - The diagram represents BN occurring at a *single neuron*
- Let’s complete the figure and work out the derivatives

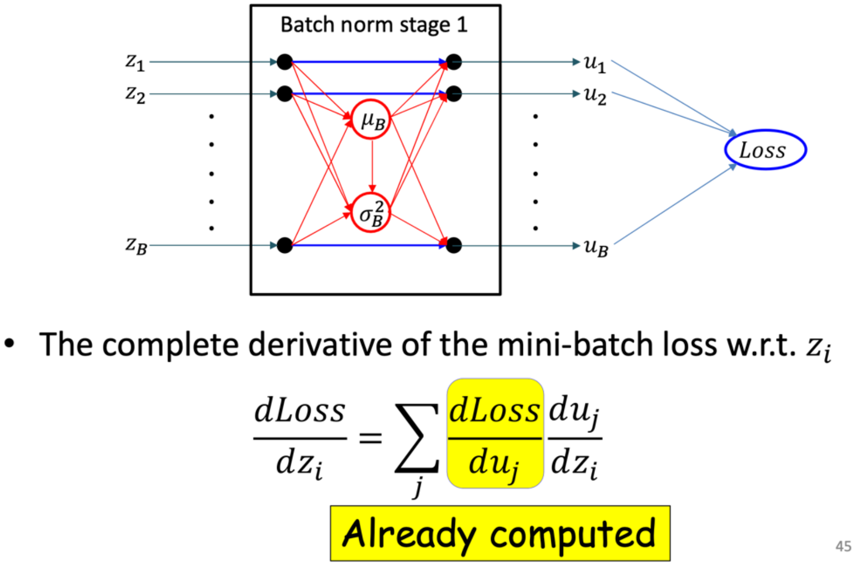
Batch Normalization -- Backpropagation



- The complete derivative of the mini-batch loss w.r.t. z_i

$$\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j} \frac{du_j}{dz_i}$$

Batch Normalization -- Backpropagation



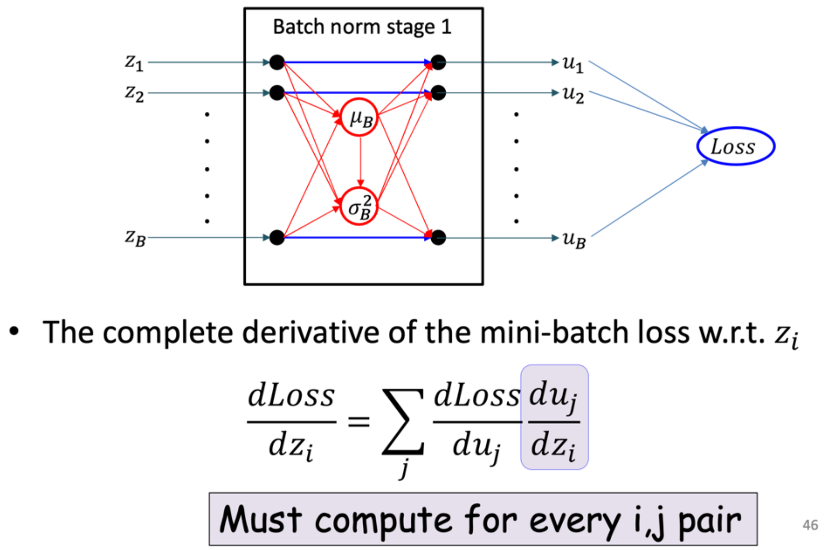
- The complete derivative of the mini-batch loss w.r.t. z_i

$$\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j}\frac{du_j}{dz_i}$$

Already computed

45

Batch Normalization -- Backpropagation



- The complete derivative of the mini-batch loss w.r.t. z_i

$$\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j} \boxed{\frac{du_j}{dz_i}}$$

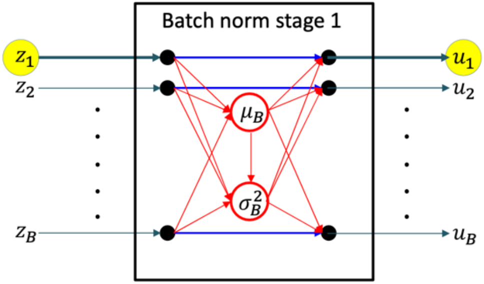
Must compute for every i,j pair

Batch Normalization -- Backpropagation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- The derivative for the "through" line ($i = j$)

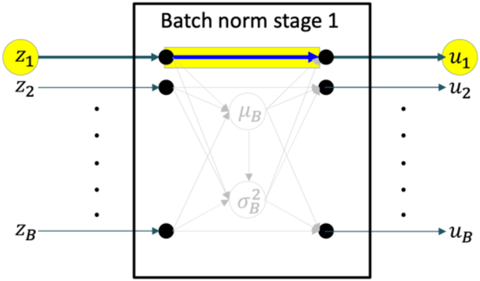
$$\frac{du_i}{dz_i} =$$

Batch Normalization -- Backpropagation

$$\boxed{\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i}$$

$$\boxed{\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2}$$

$$\boxed{u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}}$$



- The derivative for the “through” line ($i = j$)

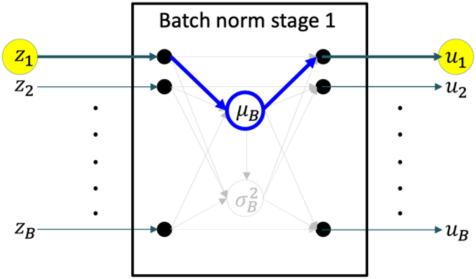
$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} +$$

Batch Normalization -- Backpropagation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- The derivative for the “through” line ($i = j$)

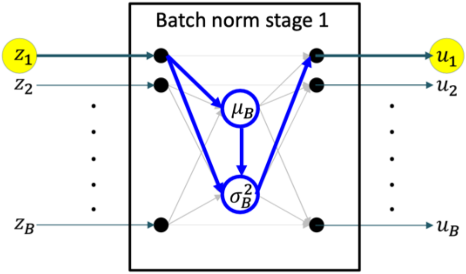
$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} +$$

Batch Normalization -- Backpropagation

$$\boxed{\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i}$$

$$\boxed{\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2}$$

$$\boxed{u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}}$$



- The derivative for the “through” line ($i = j$)

$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}$$

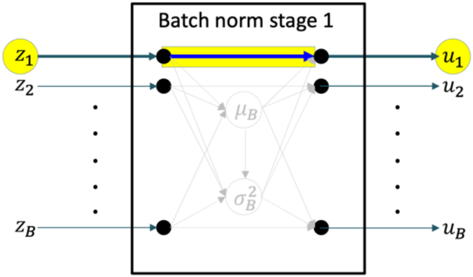
Batch Normalization -- Backpropagation

$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}$$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- From the highlighted relation

$$\frac{\partial u_i}{\partial z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}}$$

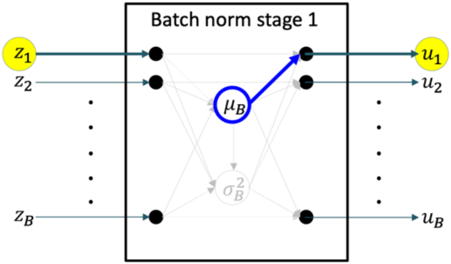
Batch Normalization -- Backpropagation

$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}$$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- From the highlighted relation

$$\frac{\partial u_i}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}}$$

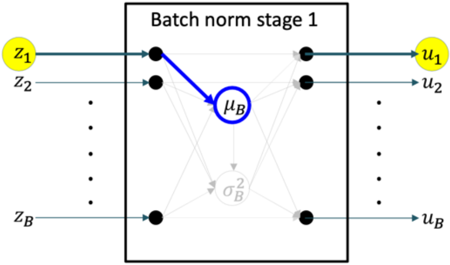
Batch Normalization -- Backpropagation

$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}$$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- From the highlighted relation

$$\frac{\partial \mu_B}{\partial z_i} = \frac{1}{B}$$

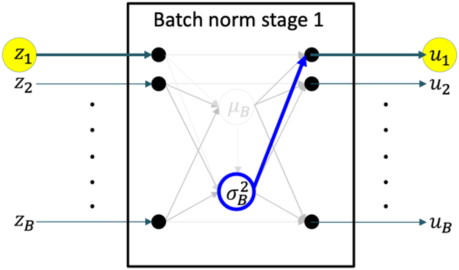
Batch Normalization -- Backpropagation

$$\boxed{\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}}$$

$$\boxed{\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i}$$

$$\boxed{\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2}$$

$$\boxed{u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}}$$



- From the highlighted equation

$$\frac{\partial u_i}{\partial \sigma_B^2} = \frac{-(z_i - \mu_B)}{2}(\sigma_B^2 + \epsilon)^{-3/2}$$

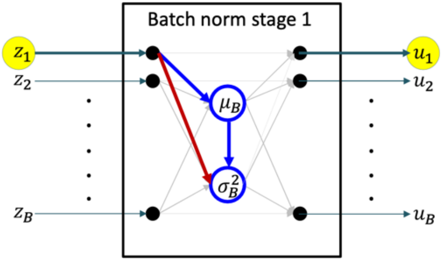
Batch Normalization -- Backpropagation

$$\boxed{\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}}$$

$$\boxed{\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i}$$

$$\boxed{\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2}$$

$$\boxed{u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}}$$



- From the highlighted equations

$$\frac{d\sigma_B^2}{dz_i} = \frac{\partial \sigma_B^2}{\partial z_i} + \frac{\partial \sigma_B^2}{\partial \mu_B}\frac{d\mu_B}{dz_i}$$

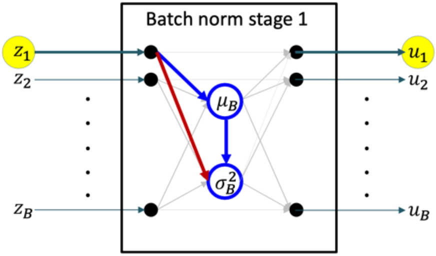
Batch Normalization -- Backpropagation

$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}$$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- From the highlighted equations

$$\frac{d\sigma_B^2}{dz_i} = \frac{2(z_i - \mu_B)}{B}$$

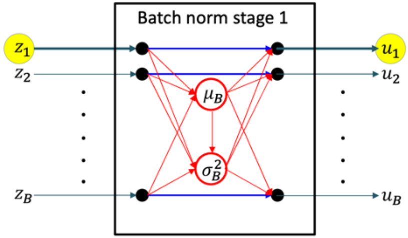
Batch Normalization -- Backpropagation

$$\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B}\frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2}\frac{d\sigma_B^2}{dz_i}$$

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- **The derivative for the "through" line ($i = j$)**

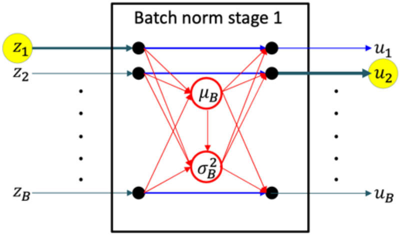
$$\frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}}$$

Batch Normalization -- Backpropagation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

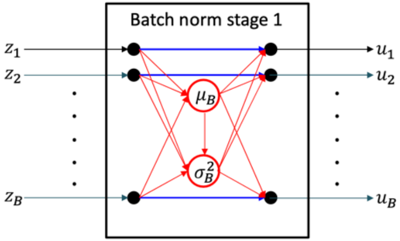
$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



- The derivative for the "cross" lines ($i \neq j$)

$$\frac{du_j}{dz_i} = \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}}$$

Batch Normalization -- Backpropagation



$$\frac{du_j}{dz_i} = \begin{cases} \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}} & \text{if } j = i \\ \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}} & \text{if } j \neq i \end{cases}$$

Batch Normalization -- Backpropagation

$$\frac{du_j}{dz_i} = \begin{cases} \frac{1}{\sqrt{\sigma_B^2+\epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2+\epsilon}} + \frac{-(z_i-\mu_B)^2}{B(\sigma_B^2+\epsilon)^{3/2}} & \text{if } j = i \\ \frac{-1}{B\sqrt{\sigma_B^2+\epsilon}} + \frac{-(z_i-\mu_B)^2}{B(\sigma_B^2+\epsilon)^{3/2}} & \text{if } j \neq i \end{cases}$$

$$\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j}\frac{du_j}{dz_i}$$

- The complete derivative of the mini-batch loss w.r.t. z_i

$$\frac{dLoss}{dz_i} = \frac{1}{\sqrt{\sigma_B^2+\epsilon}}\frac{dLoss}{du_i} - \frac{1}{B\sqrt{\sigma_B^2+\epsilon}}\sum_j \frac{dLoss}{du_j} - \frac{1}{B(\sigma_B^2+\epsilon)^{3/2}}\sum_j \frac{dLoss}{du_j}(z_i-\mu_B)^2$$

Batch Normalization -- Backpropagation

$$\frac{du_j}{dz_i} = \begin{cases} \frac{1}{\sqrt{\sigma_B^2+\epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2+\epsilon}} + \frac{-(z_i-\mu_B)^2}{B(\sigma_B^2+\epsilon)^{3/2}} & \text{if } j = i \\ \frac{-1}{B\sqrt{\sigma_B^2+\epsilon}} + \frac{-(z_i-\mu_B)^2}{B(\sigma_B^2+\epsilon)^{3/2}} & \text{if } j \neq i \end{cases}$$

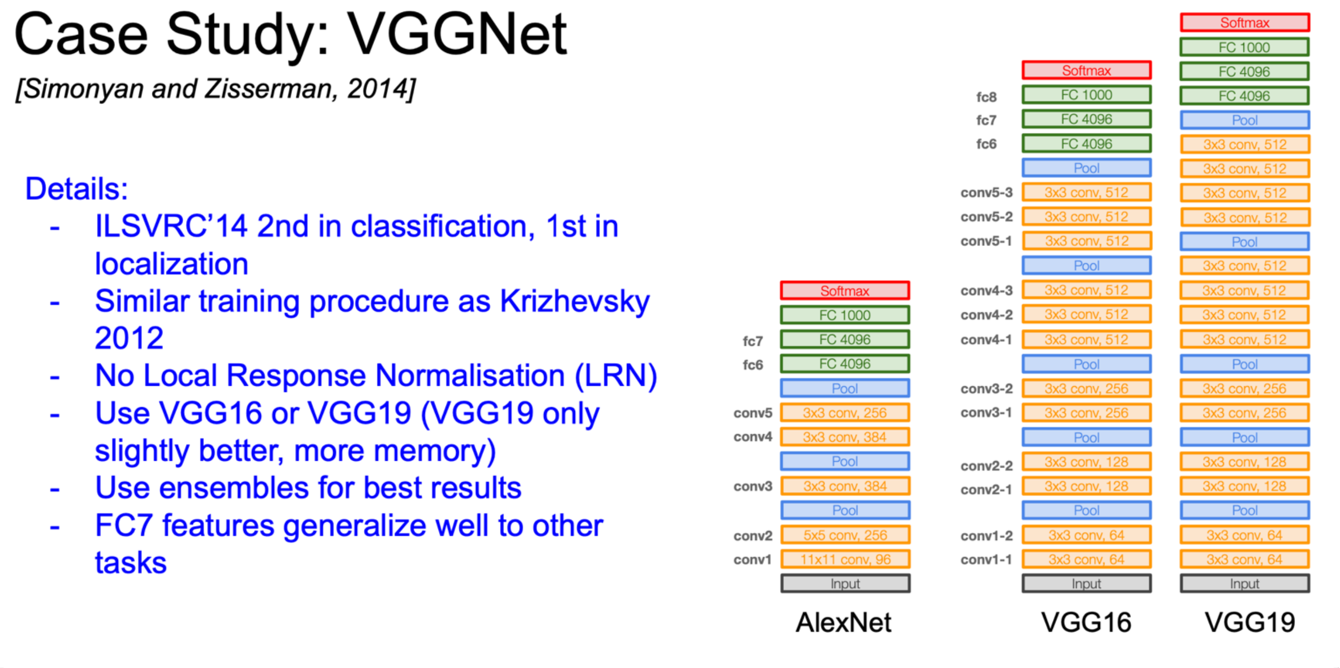
$$\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j}\frac{du_j}{dz_i}$$

- The complete derivative of the mini-batch loss w.r.t. z_i

$$\frac{dLoss}{dz_i} = \frac{1}{\sqrt{\sigma_B^2+\epsilon}}\frac{dLoss}{du_i} - \frac{1}{B\sqrt{\sigma_B^2+\epsilon}}\sum_j \frac{dLoss}{du_j} - \frac{1}{B(\sigma_B^2+\epsilon)^{3/2}}\sum_j \frac{dLoss}{du_j}(z_i-\mu_B)^2$$

After Break: CNN Architectures

VGGNet



Case Study: VGGNet

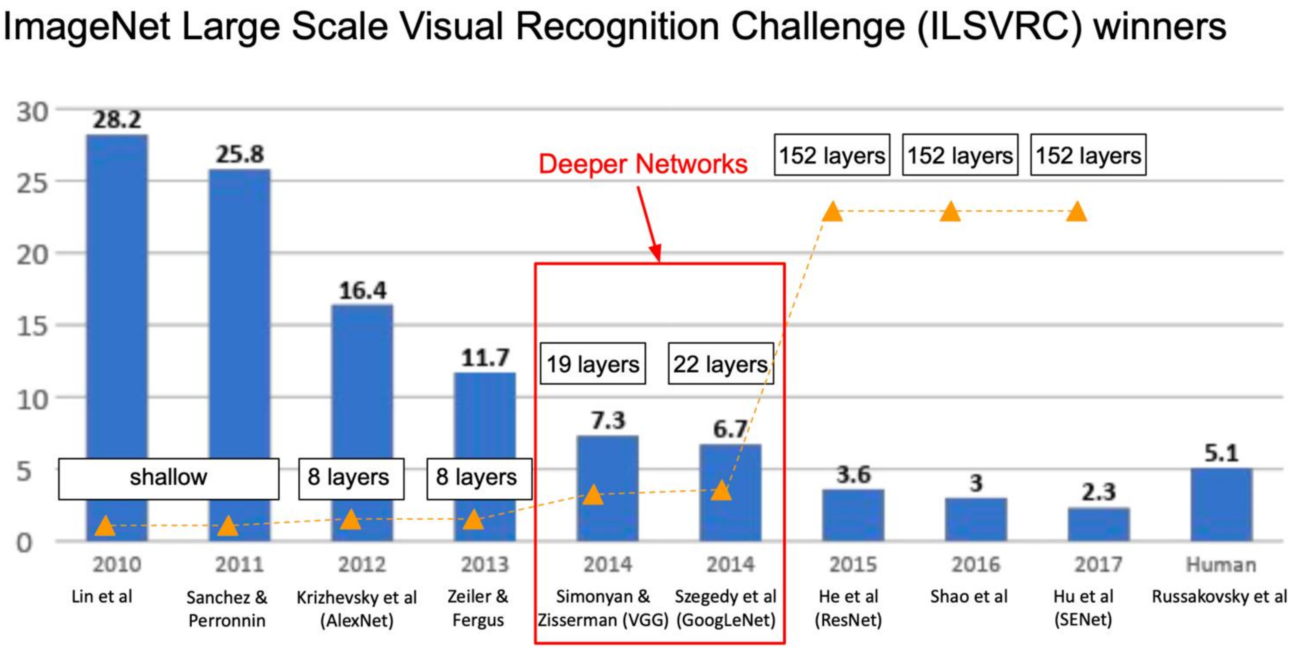
[Simonyan and Zisserman, 2014]

Details:

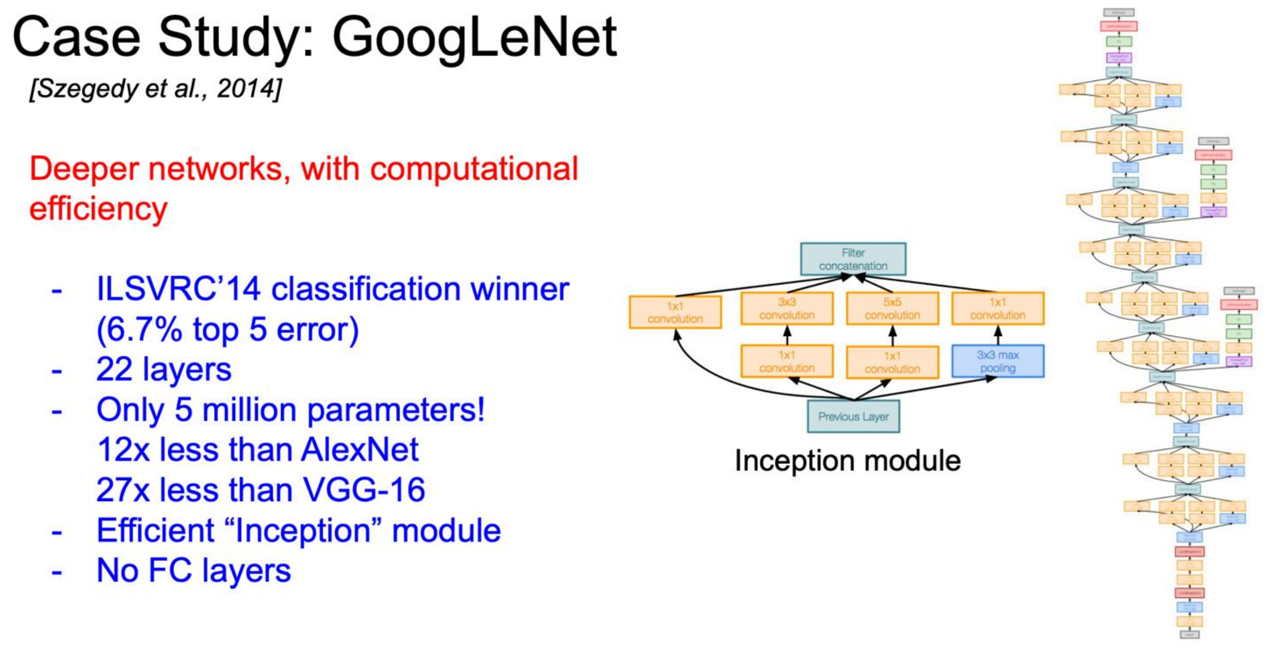
- ILSVRC’14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

Source: https://cs231n.stanford.edu/

Deeper Networks



GoogLeNet



Case Study: GoogLeNet

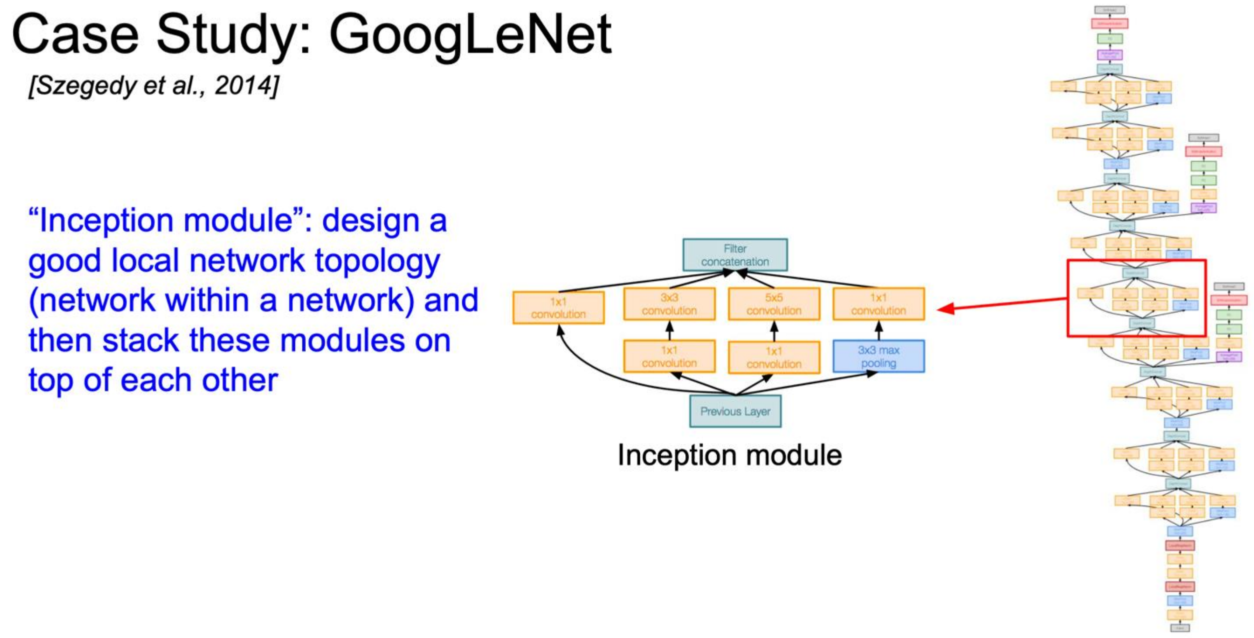
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient "Inception" module
- No FC layers

Inception module

GoogLeNet



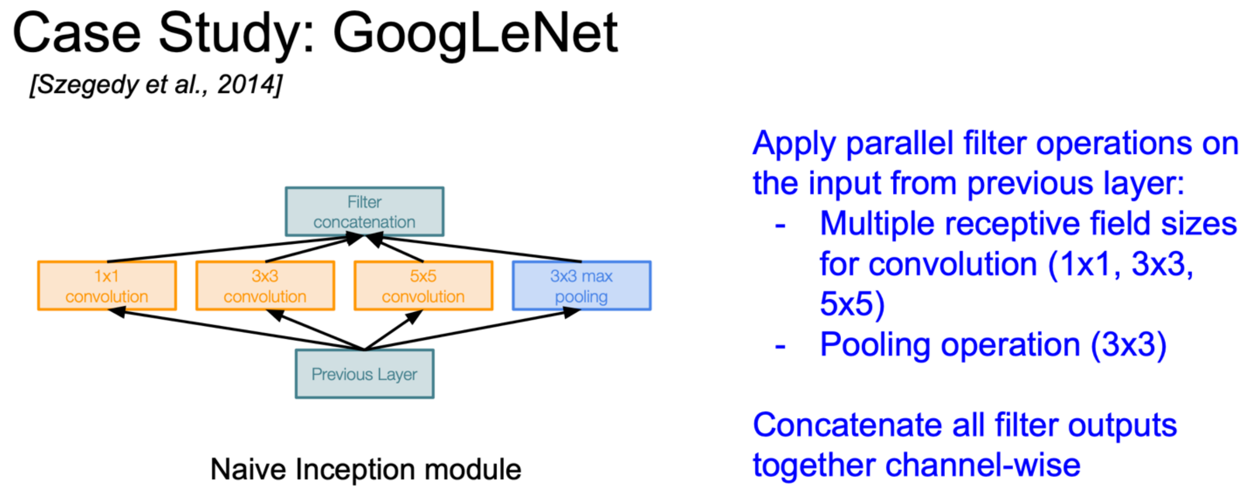
Case Study: GoogLeNet

[Szegedy et al., 2014]

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other

Inception module

GoogLeNet



Case Study: GoogLeNet

[Szegedy et al., 2014]

Naive Inception module

Apply parallel filter operations on the input from previous layer:

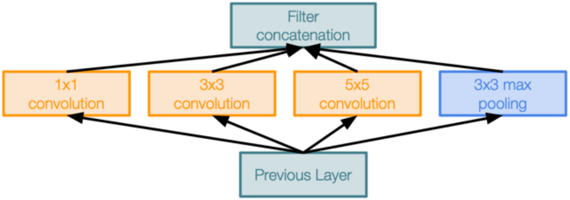
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

Q: What is the problem with this?
[Hint: Computational complexity]

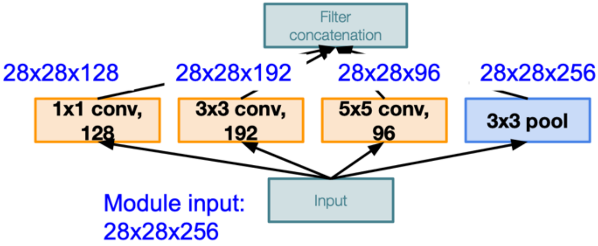
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q1: What are the output sizes of all different filter operations?



Filter concatenation

28x28x128

28x28x192

28x28x96

28x28x256

1x1 conv, 128

3x3 conv, 192

5x5 conv, 96

3x3 pool

Input

Module input: 28x28x256

Naive Inception module

Q: What is the problem with this? [Hint: Computational complexity]

GoogLeNet

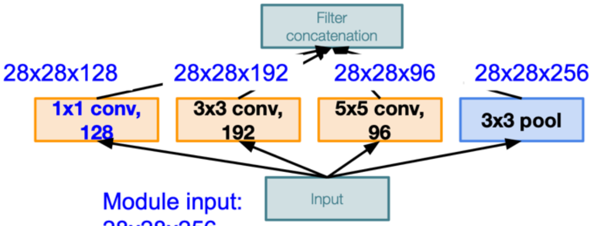
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2:What is output size after filter concatenation?

28x28x(128+192+96+256) = 28x28x672



Module input: 28x28x256

Naive Inception module

Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x**192x3x3x256**
[5x5 conv, 96] 28x28x**96x5x5x256**
Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

GoogLeNet

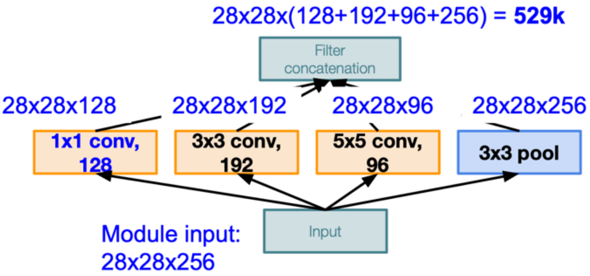
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q2:What is output size after filter concatenation?



28x28x(128+192+96+256) = **529k**

Filter concatenation

28x28x128

28x28x192

28x28x96

28x28x256

1x1 conv, 128

3x3 conv, 192

5x5 conv, 96

3x3 pool

Input

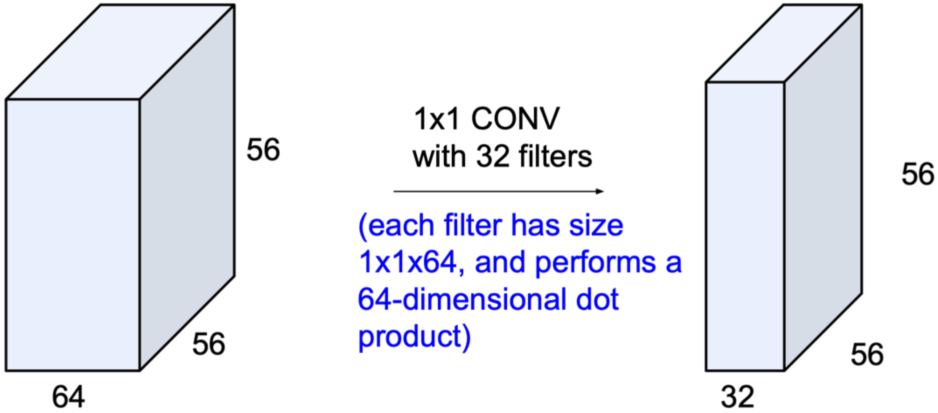
Module input: 28x28x256

Naive Inception module

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature channel size

GoogLeNet

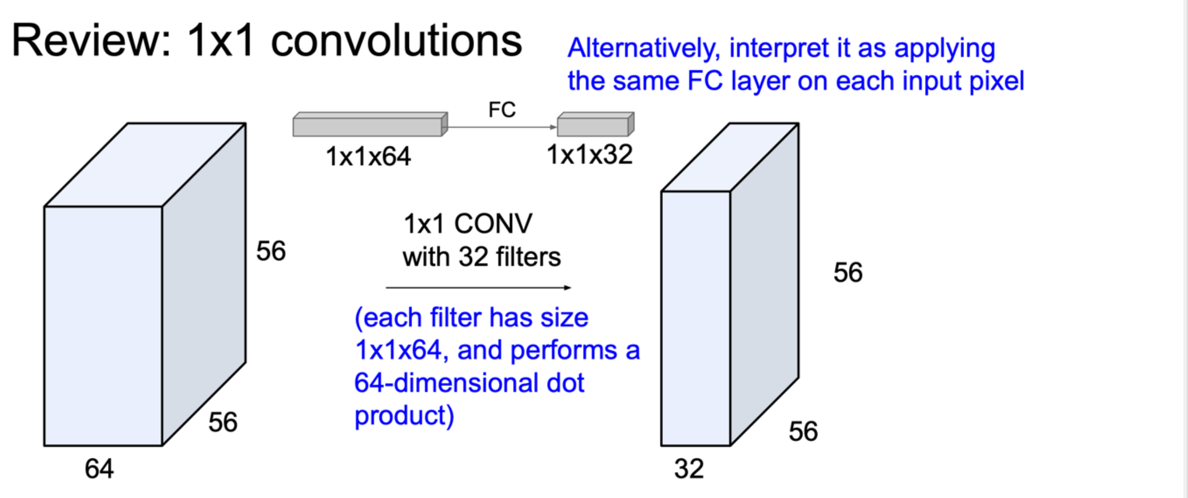
Review: 1x1 convolutions



1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

GoogLeNet

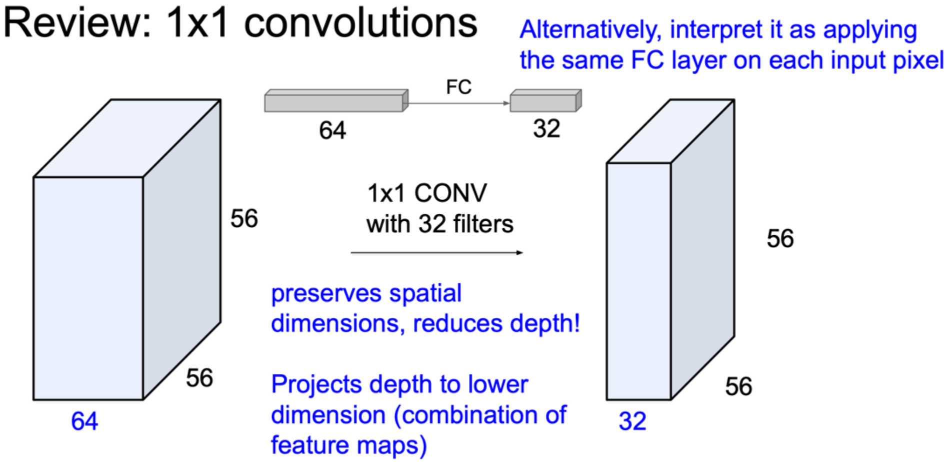


Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

GoogLeNet



Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel

1x1 CONV with 32 filters

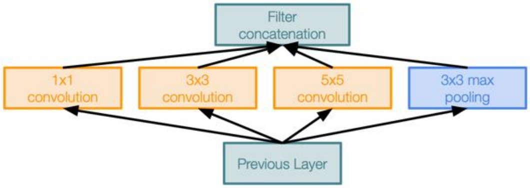
preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)

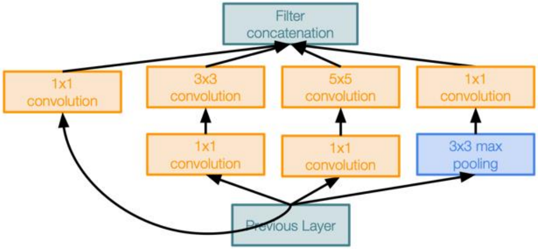
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

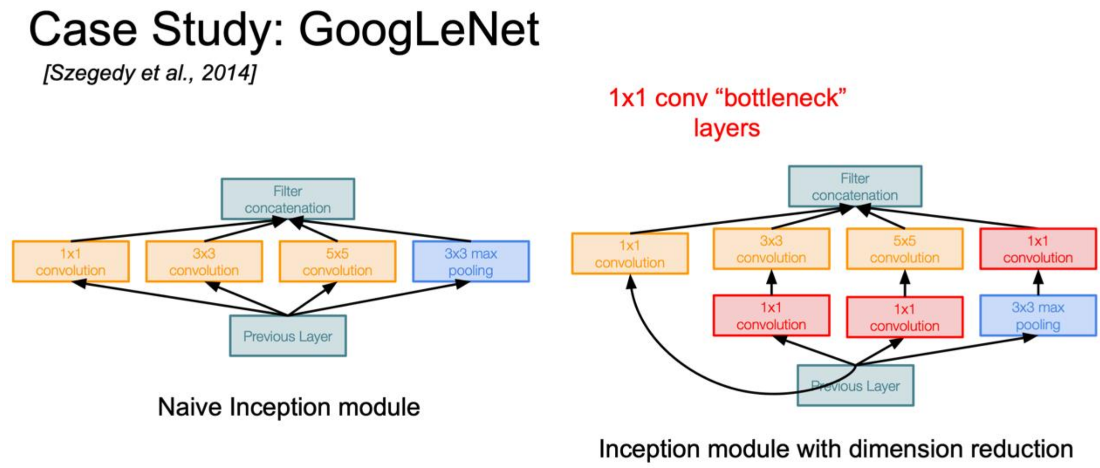


Naive Inception module



Inception module with dimension reduction

GoogLeNet



Case Study: GoogLeNet

[Szegedy et al., 2014]

1x1 conv "bottleneck" layers

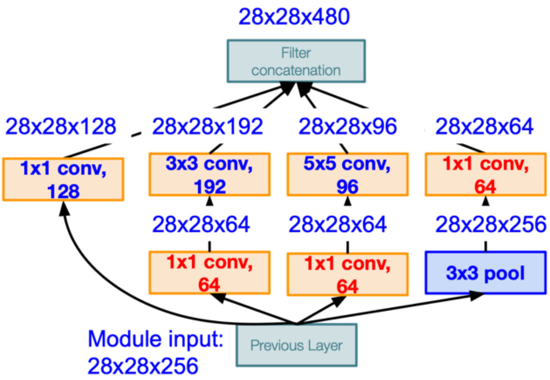
Naive Inception module

Inception module with dimension reduction

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

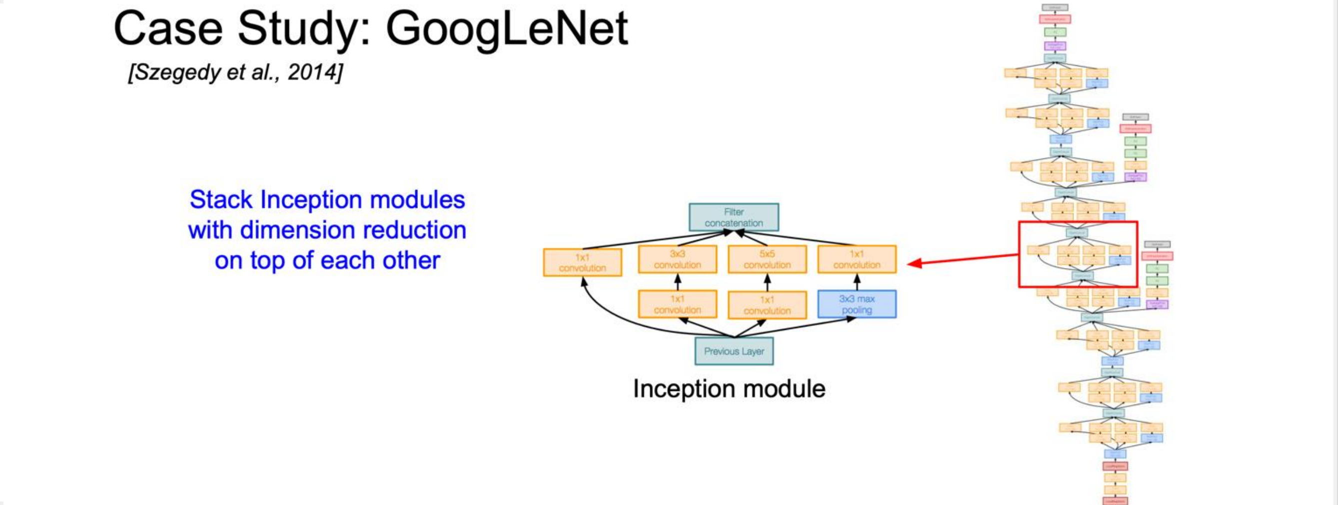
Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

GoogLeNet



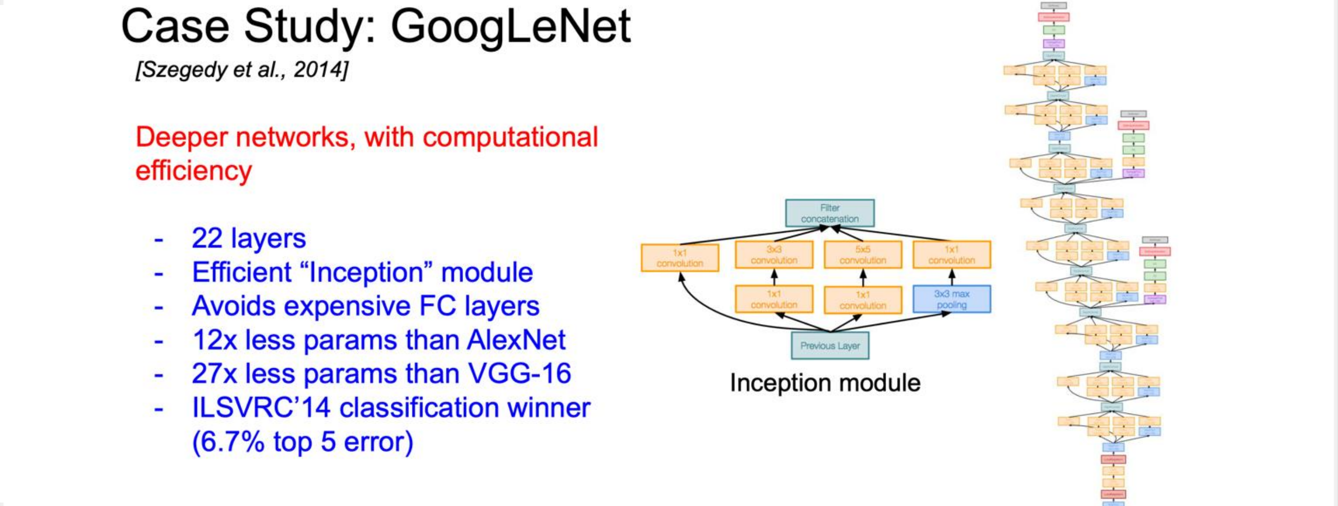
Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules
with dimension reduction
on top of each other

Inception module

GoogLeNet



Case Study: GoogLeNet

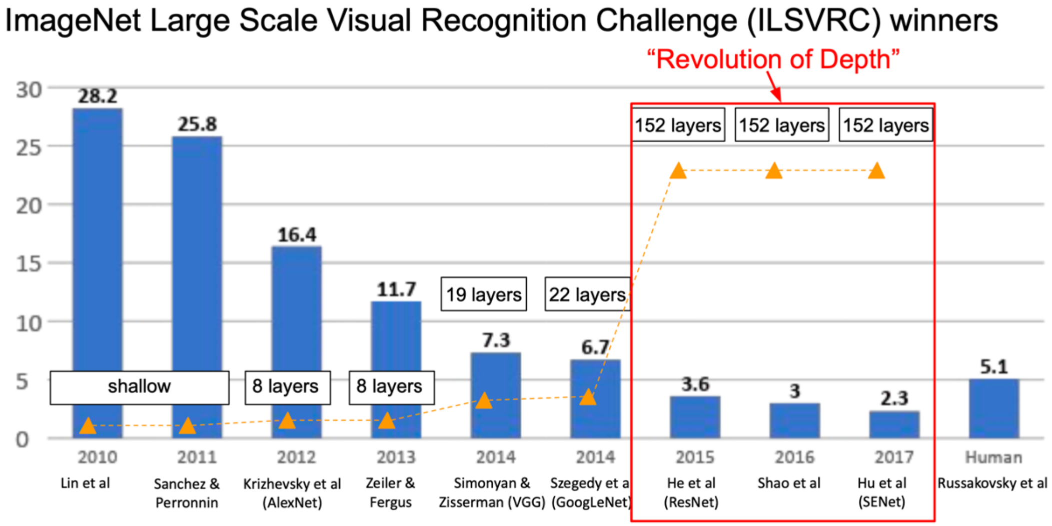
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

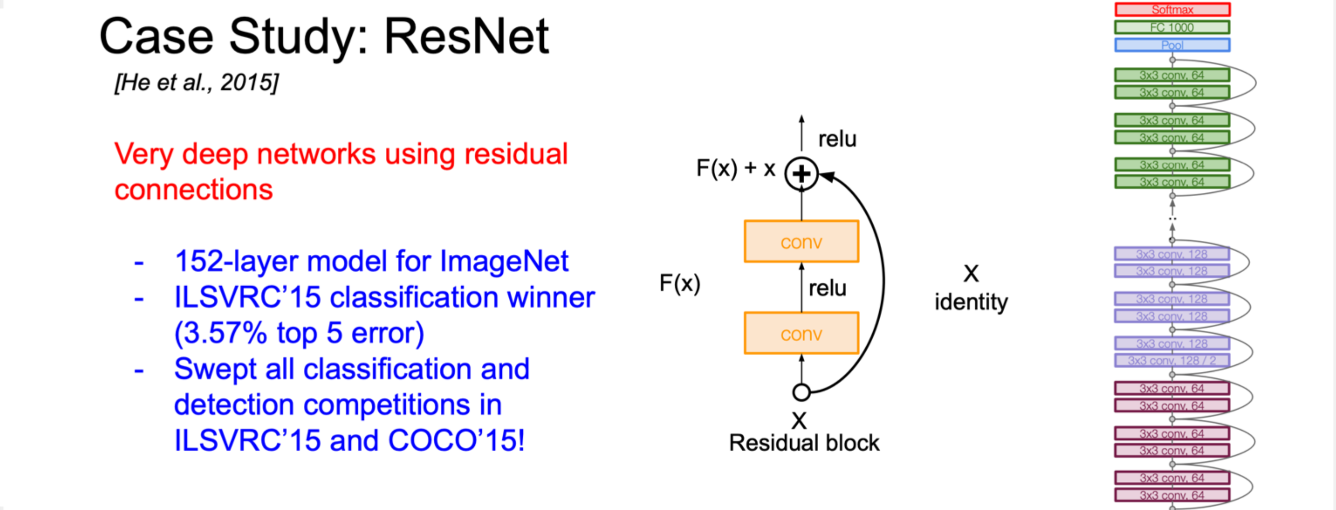
- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)

Inception module

ResNet


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
"Revolution of Depth"
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)
2014
Szegedy et al (GoogLeNet)
2015
He et al (ResNet)
2016
Shao et al
2017
Hu et al (SENet)
Human
Russakovsky et al

ResNet



Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

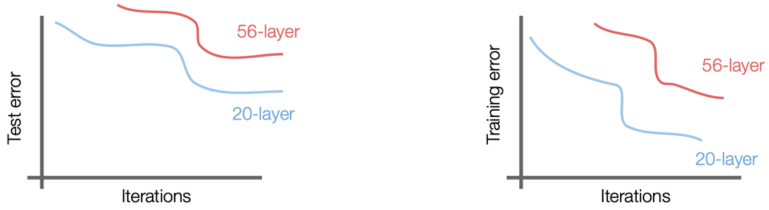
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

ResNet

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

ResNet

Case Study: ResNet

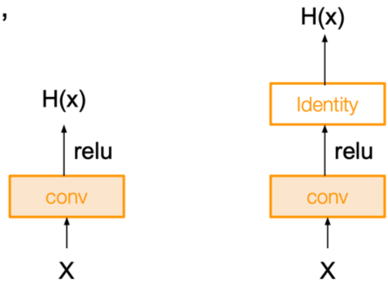
[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

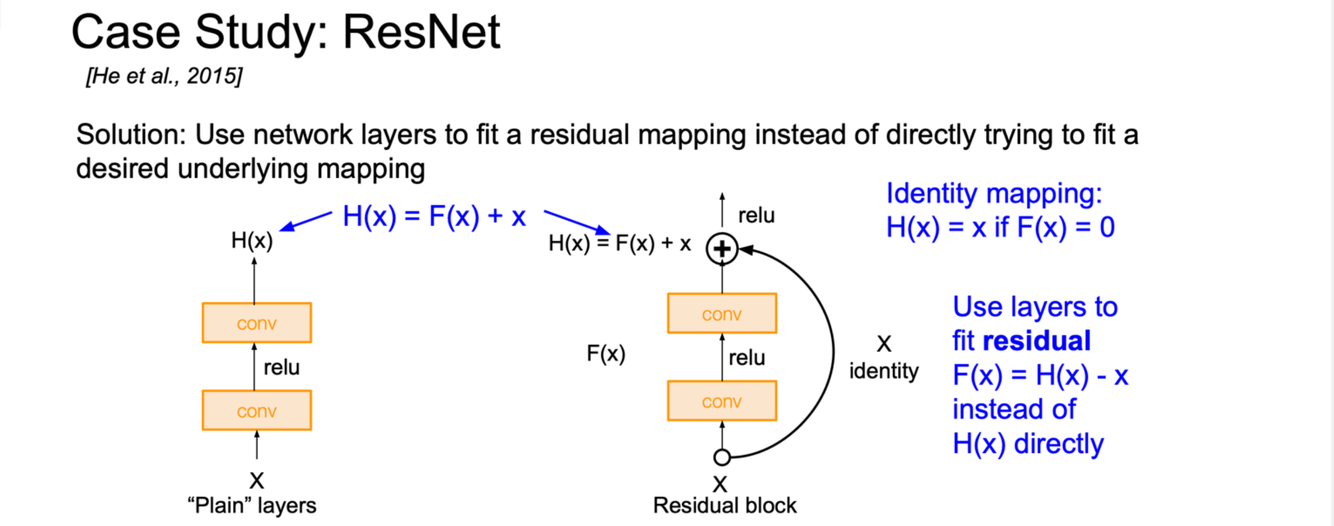
Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



ResNet



Case Study: ResNet

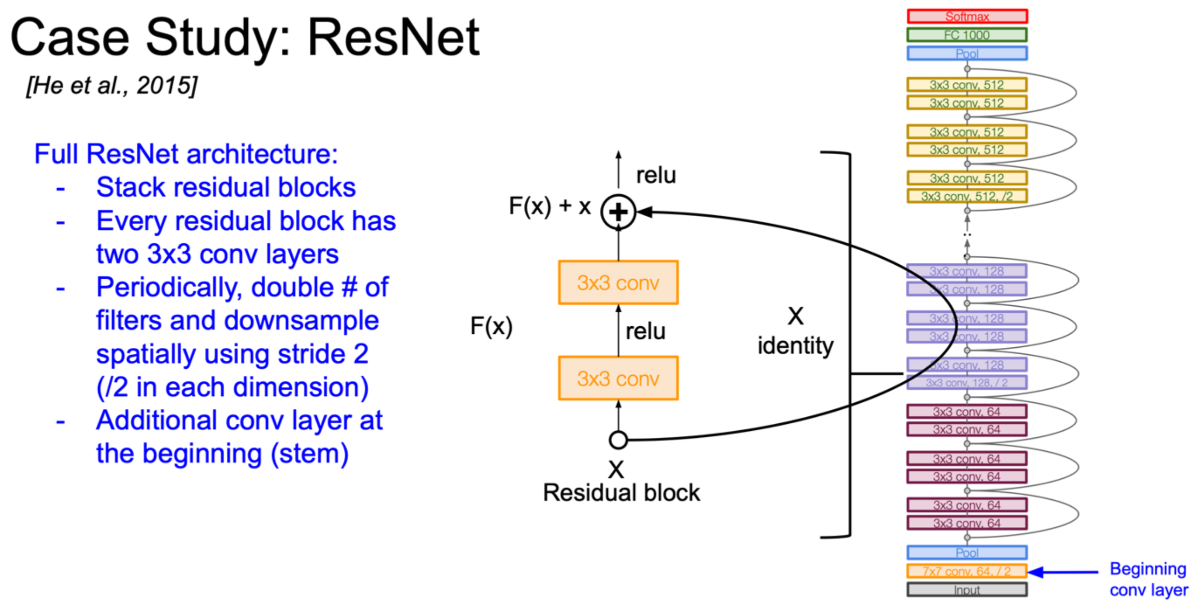
[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

Identity mapping:
$H(x) = x$ if $F(x) = 0$

Use layers to fit **residual** $F(x) = H(x) - x$ instead of $H(x)$ directly

ResNet



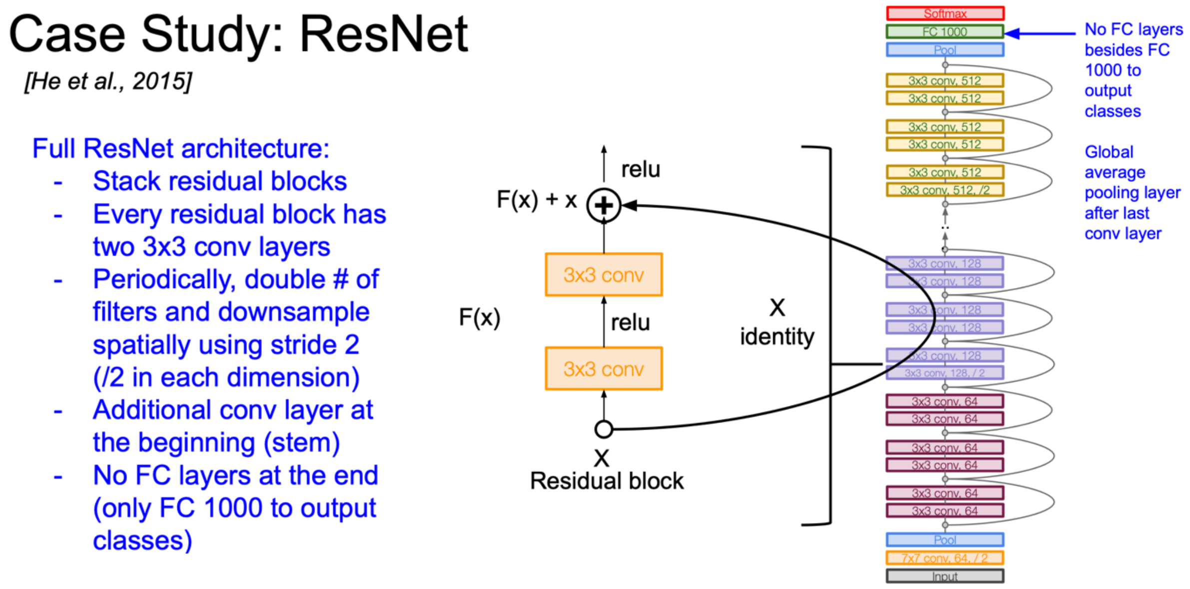
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)

Beginning conv layer

ResNet



Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)

No FC layers besides FC 1000 to output classes

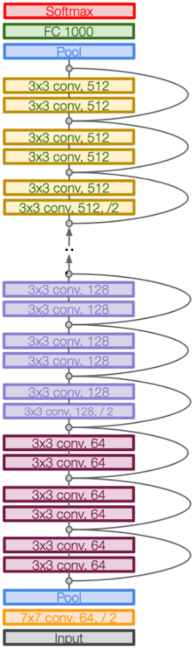
Global average pooling layer after last conv layer

ResNet

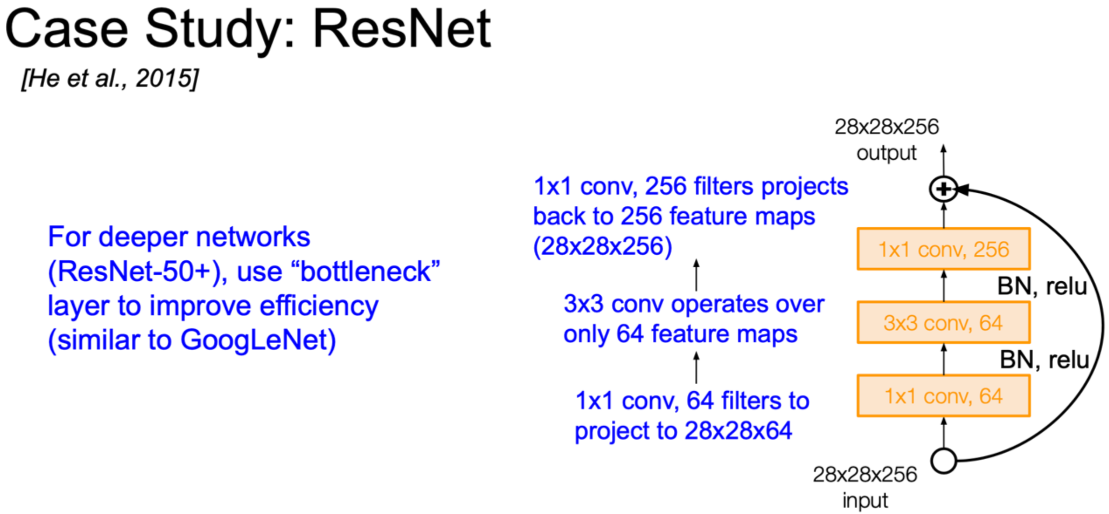
Case Study: ResNet

[He et al., 2015]

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet



ResNet



Case Study: ResNet

[He et al., 2015]

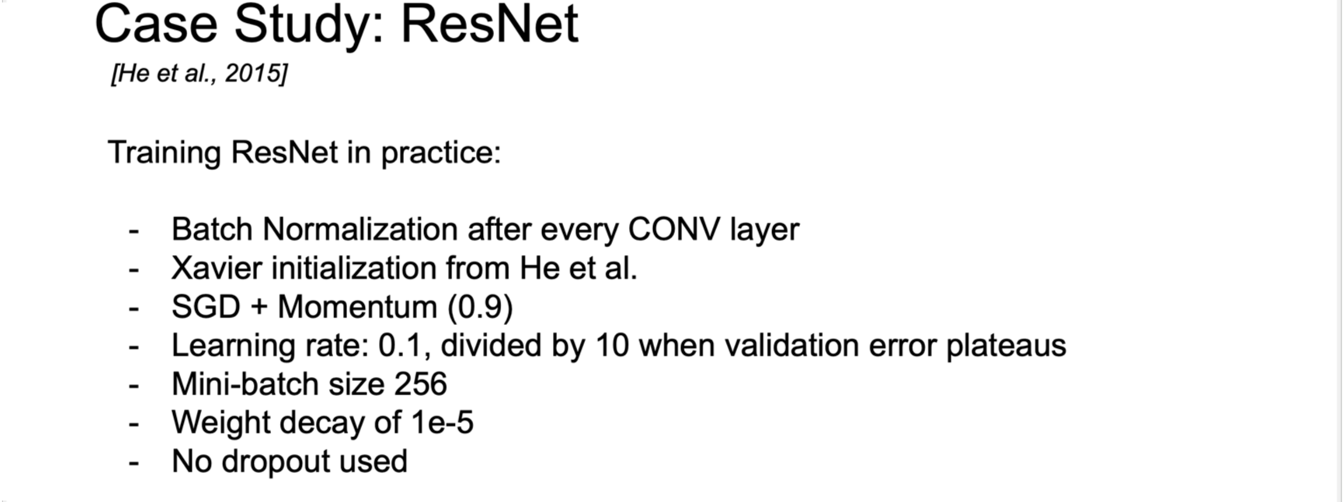
For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64

ResNet



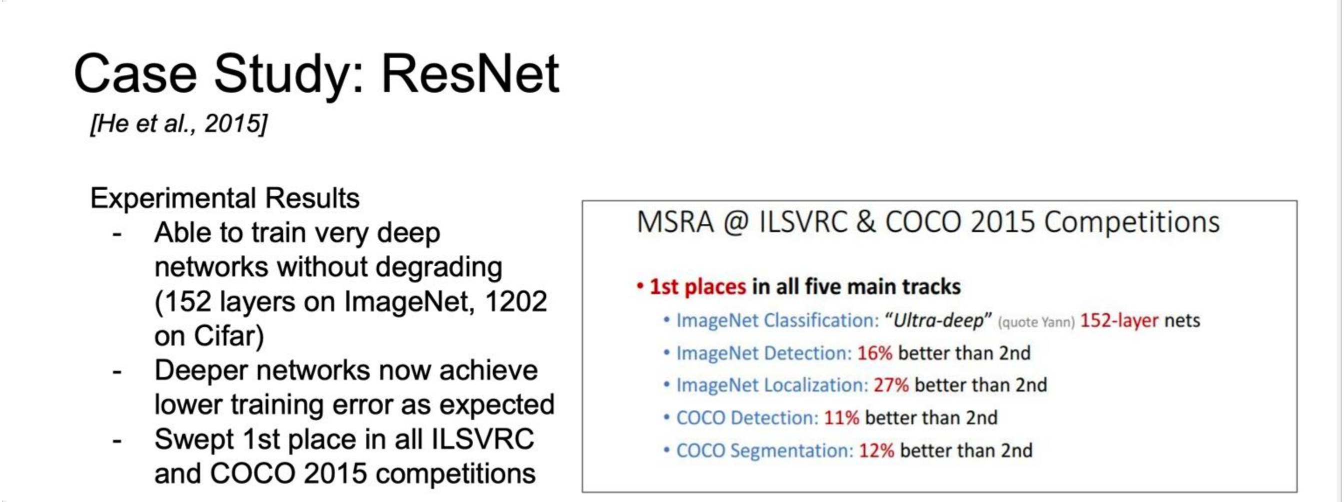
Case Study: ResNet

[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

ResNet



Case Study: ResNet

[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

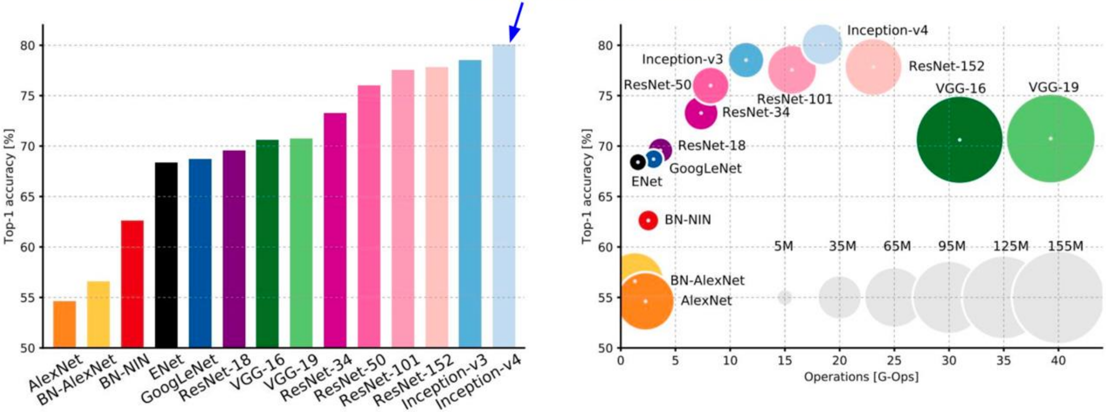
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *“Ultra-deep”* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

Improving ResNet

Comparing complexity...

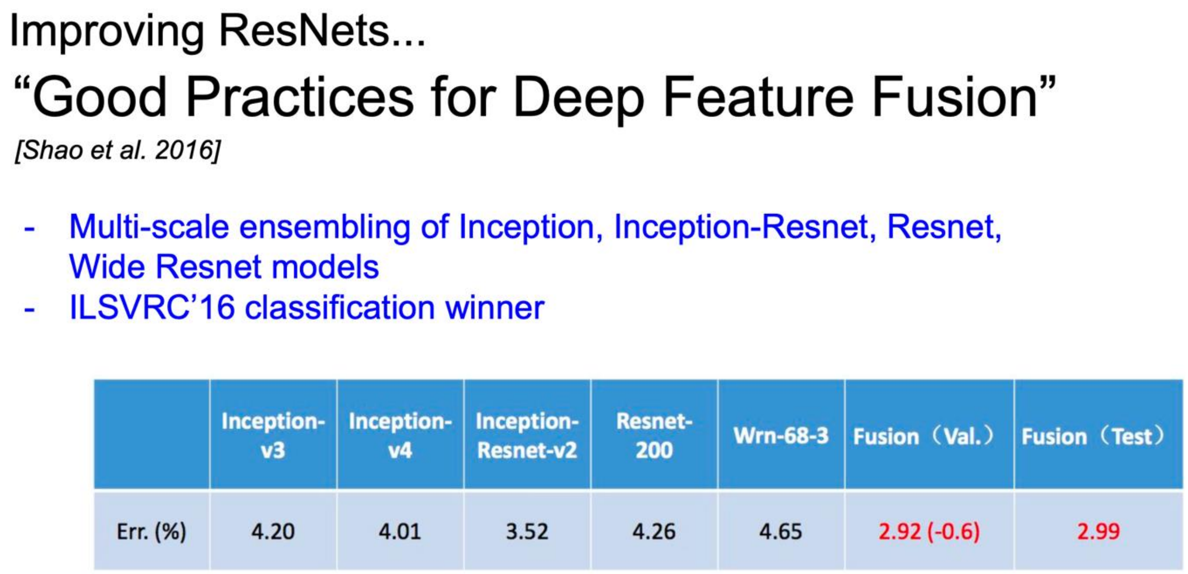
Inception-v4: Resnet + Inception!



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Improving ResNet



Improving ResNets...

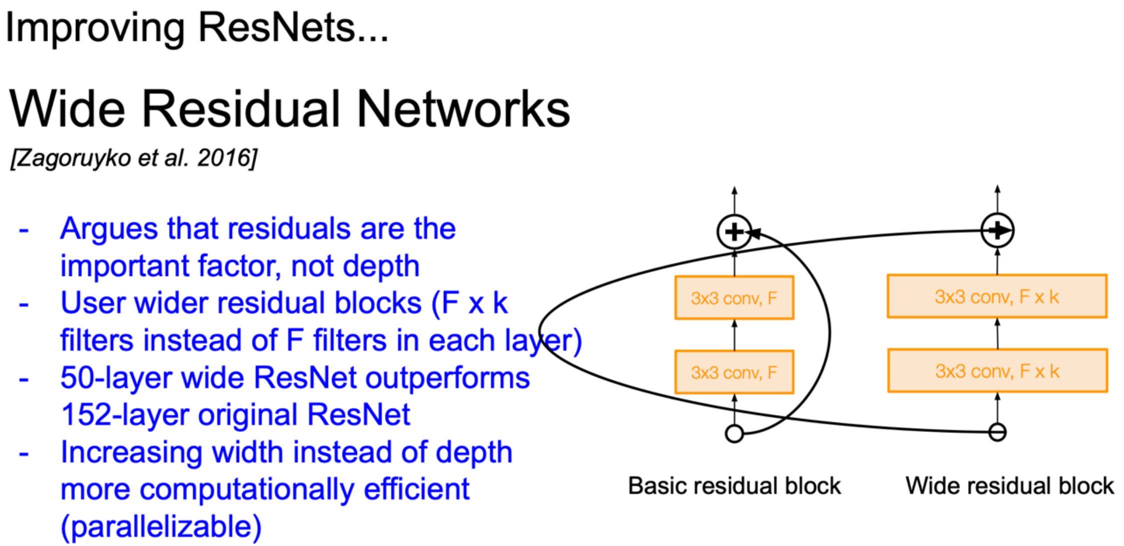
“Good Practices for Deep Feature Fusion”

[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

	Inception-v3	**Inception-v4**	**Inception-Resnet-v2**	**Resnet-200**	**Wrn-68-3**	**Fusion（Val.）**	**Fusion（Test）**
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

Improving ResNet



Improving ResNets...

Wide Residual Networks

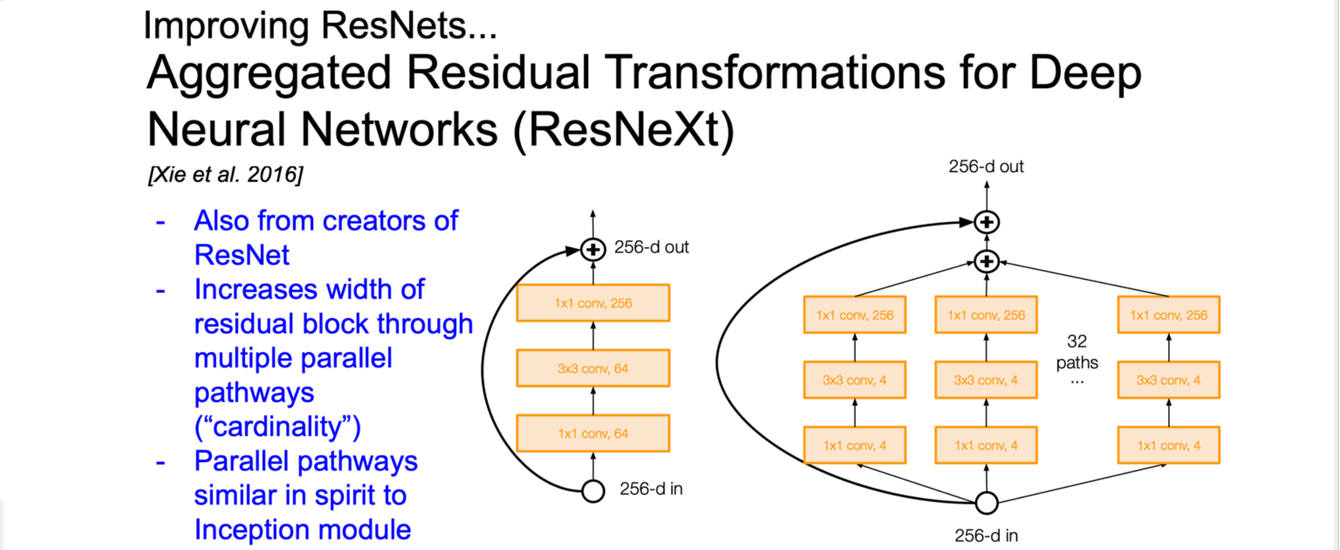
[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)

Basic residual block

Wide residual block

Improving ResNet



Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module

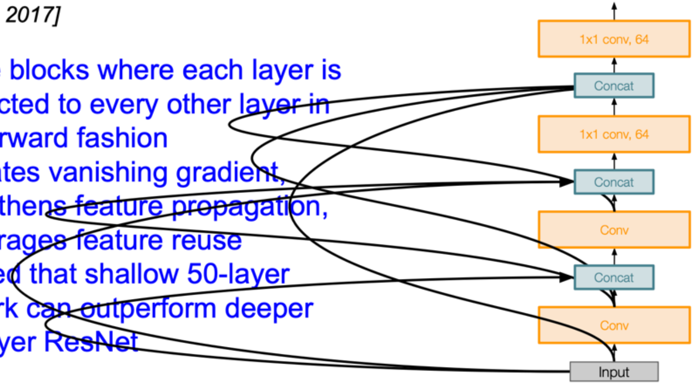
Improving ResNet

Other ideas...

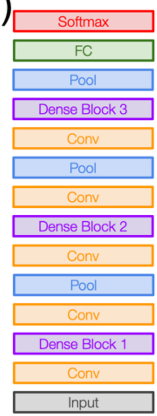
Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



Dense Block



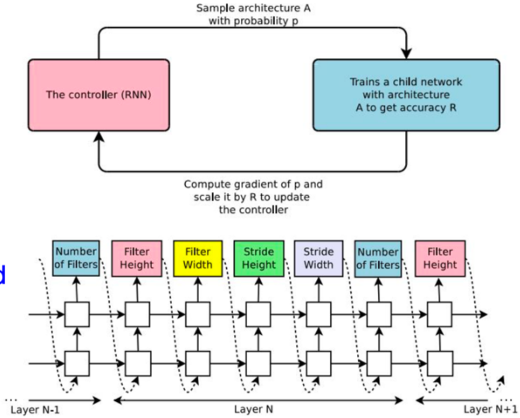
Improving ResNet

Learning to search for network architectures...

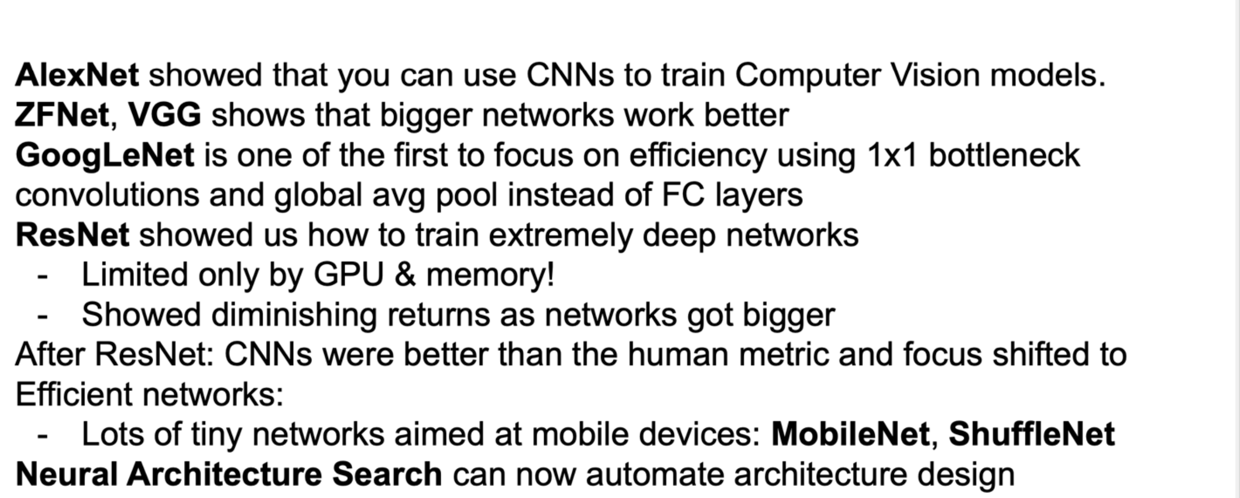
Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- “Controller” network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a “reward” R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)



Main Take-Aways



AlexNet showed that you can use CNNs to train Computer Vision models.

ZFNet, **VGG** shows that bigger networks work better

GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers

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 Efficient networks:

- Lots of tiny networks aimed at mobile devices: **MobileNet**, **ShuffleNet**

Neural Architecture Search can now automate architecture design