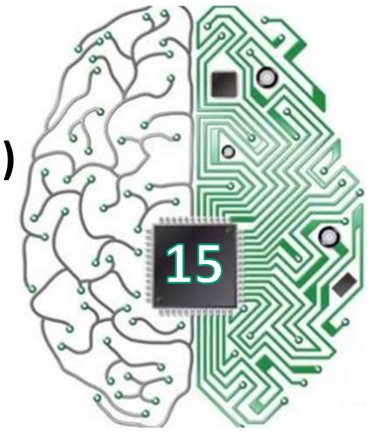
Open Elective Course [OE]
Course Code: CSO507
Winter 2023-24

Lecture#

Deep Learning

Unit-4: Convolutional Neural Networks (Part-III)



Course Instructor:
Dr. Monidipa Das
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology (Indian School of Mines) Dhanbad, Jharkhand 826004, India

Convolution Example (revisited)

2

Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ? (32+2*2-5)/1+1 = 32 spatially, so
10 x 32 x 32

Number of learnable parameters: ? Parameters per filter: 3*5*5 + 1 (for bias) = 76
10 filters, so total is 10 * 76 = **760**

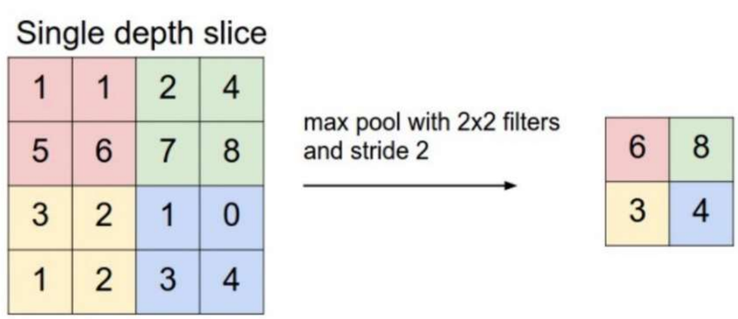
Number of multiply-add operations: ?

10*32*32 = 10,240 outputs; each output is the inner product
of two 3x5x5 tensors (75 elems); total = 75*10240 = **768K**

Pooling Layer

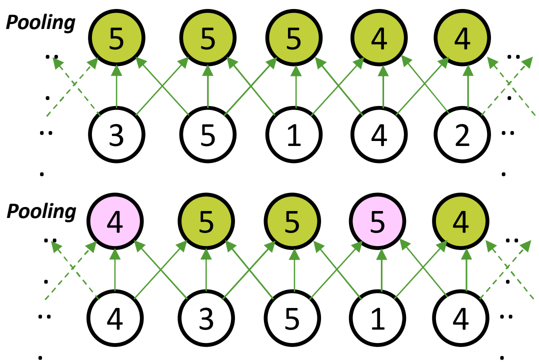
Pooling Layer: Summarizes Neighborhood

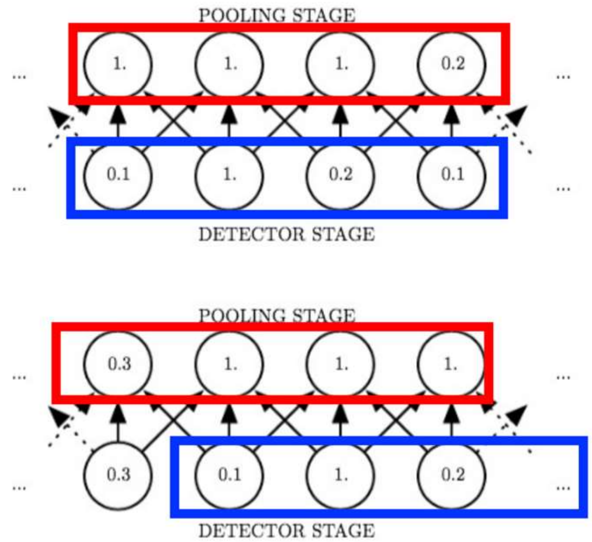
- **Max-pooling:** Applying the filter and finding the maximum value for each chunk



Pooling Layer

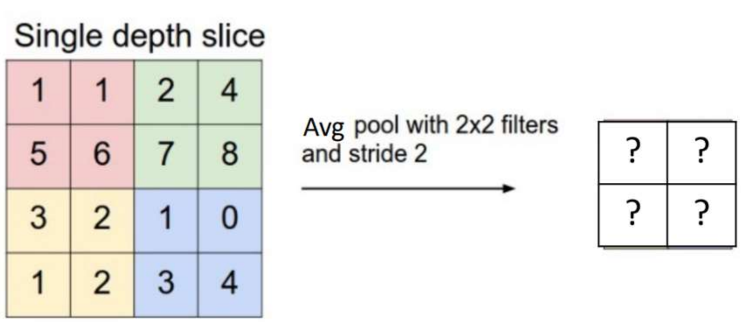
- Resilient to small translations
  - e.g.,
    - Input: all values change (shift right)
    - Output: only half the values change





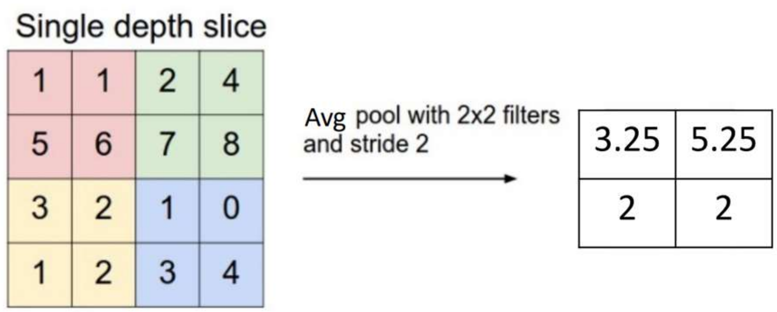
Pooling Layer: Summarizes Neighborhood

- **Max-pooling:** Applying the filter and finding the maximum value for each chunk
- **Average-pooling:** Applying the filter and finding the average value for each chunk



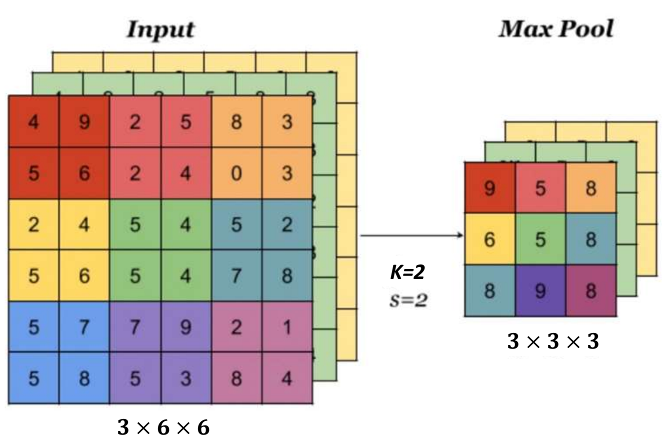
Pooling Layer: Summarizes Neighborhood

- **Max-pooling:** Applying the filter and finding the maximum value for each chunk
- **Average-pooling:** Applying the filter and finding the average value for each chunk



Pooling for Multi-Channel Input

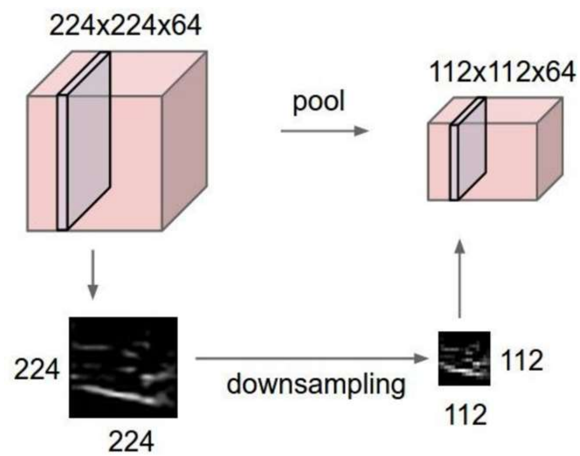
- Pooling is applied to each input channel separately



Pooling Layer: Benefits

9

- How many parameters must be learned?
  - None
- Benefits?
  - Builds in invariance to translations of the input
  - makes the representations smaller and more manageable
  - operates over each activation map independently
  - Reduces memory requirements
  - Reduces computational requirements



Pooling layer: summary

10

Input: C x H x W
Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H’ x W’ where
- H’ = (H – K) / S + 1
- W’ = (W – K) / S + 1

Learnable parameters: None!

Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

Training ConvNet

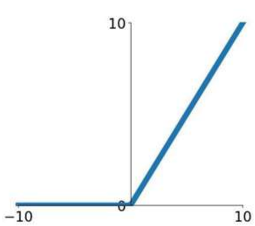
- Split and preprocess your data
- Choose your network architecture
- Initialize the weights
- Find a learning rate and regularization strength
- Minimize the loss and monitor progress

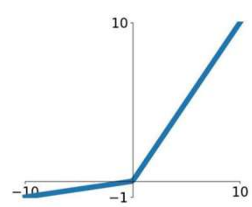
Training: Best practices

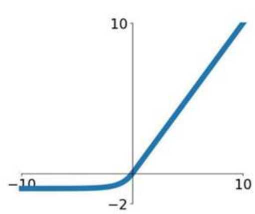
- Center (subtract mean from) your data
- Use *He, Xavier initialization* for weights
- Use RELU or leaky RELU or ELU or PReLU
- Use batch normalization
- Use data augmentation
- Use regularization
- Dropout
- Use mini-batch
- Learning rate: too high? Too low?
- Use cross-validation for hyperparameters

$w_i \sim N[0, \sigma] \quad \sigma = \sqrt{\frac{2}{fan\_in}}$

$W_{i,j}^{[l]} = \mathcal{N}\left(0, \frac{1}{n^{[l-1]}}\right)$







Data Augmentation

13

- Horizontal Flips
- Random crops and scales
- Random mix/combinations of:
  - Translation
  - Rotation
  - Stretching
  - Shearing
  - lens distortions

Regularization

14

L2 regularization $$L_{reg} = \lambda \frac{1}{2}\|W\|_2^2$$

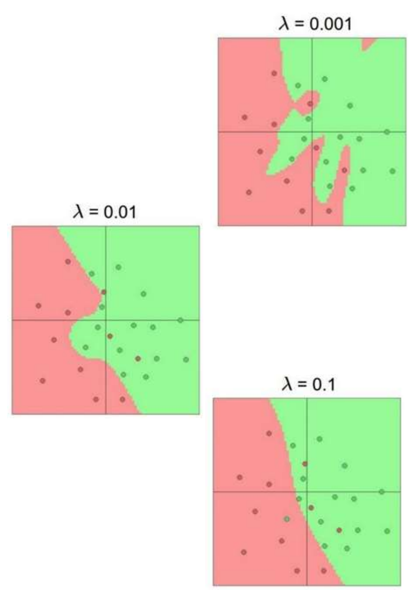
(L2 regularization encourages small weights)

L1 regularization $$L_{reg} = \lambda\|W\|_1 = \lambda \sum_{ij} |W_{ij}|$$

(L1 regularization encourages sparse weights: weights are encouraged to reduce to exactly zero)

"Elastic net" $$L_{reg} = \lambda_1\|W\|_1 + \lambda_2\|W\|_2^2$$

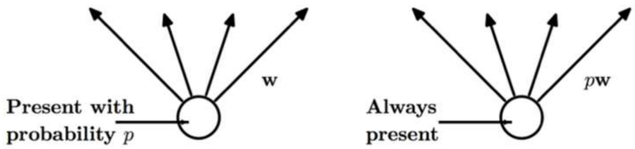
(combine L1 and L2 regularization)



Dropout

15

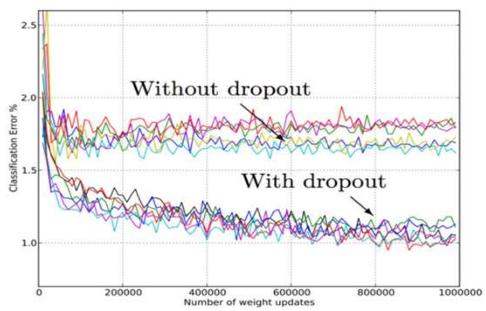
Simple but powerful technique to reduce overfitting:

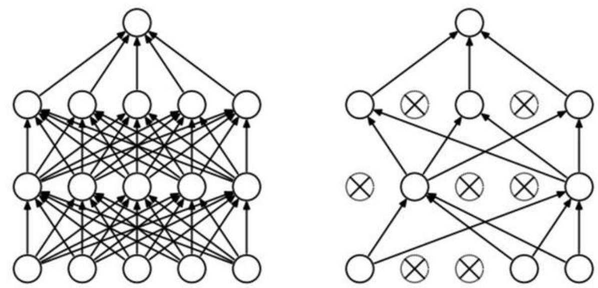


(a) At training time (b) At test time

Note: Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]





(a) Standard Neural Net (b) After applying dropout.

Mini-batch gradient descent

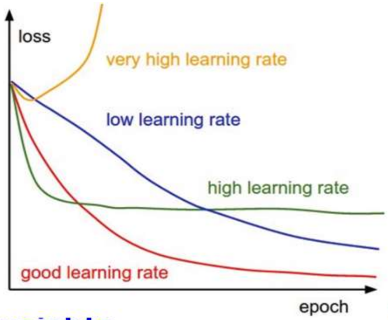
16

- In classic gradient descent, we compute the gradient from the loss for all training examples
- Could also only use some of the data for each gradient update
- We cycle through all the training examples multiple times
- Each time we've cycled through all of them once is called an 'epoch'
- Allows faster training (e.g. on GPUs), parallelization

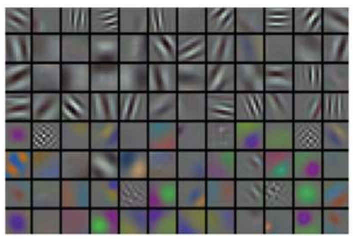
Finding a learning rate

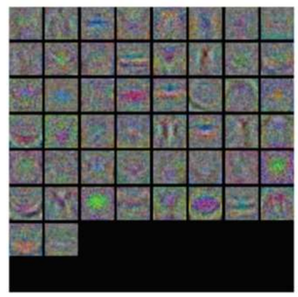
17

- **Plot the Loss**



- **Visualize the weights**

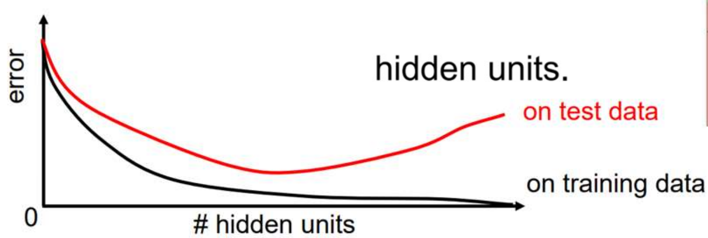


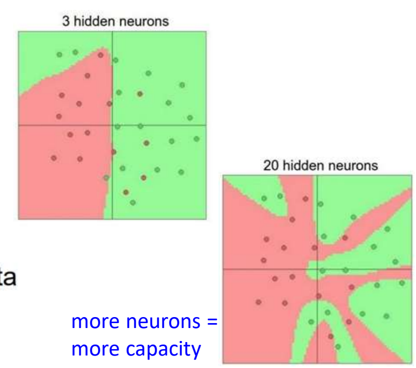


Determining best number of hidden units

18

- Too few hidden units prevent the network from adequately fitting the data.
- Too many hidden units can result in over-fitting.





- Use internal cross-validation to empirically determine an optimal number of hidden units.

Batch Normalization

19

Batch Normalization

20

- 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

21

- **Idea:** “Normalize” the inputs of a layer so they have zero mean and unit variance

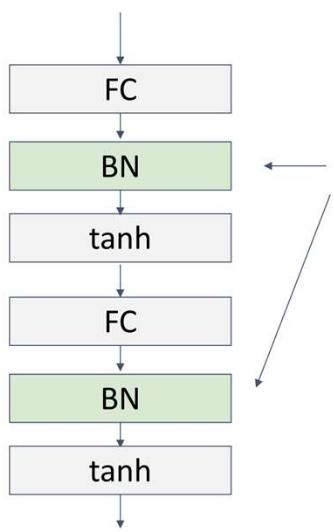
- We can normalize a batch of activations like this:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

- This is a differentiable function, so we can use it as an operator in our networks and backprop through it!

Batch Normalization

22

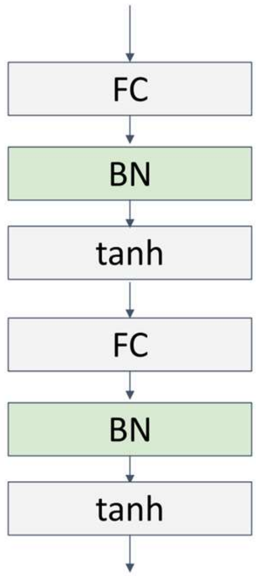


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

Batch Normalization

23

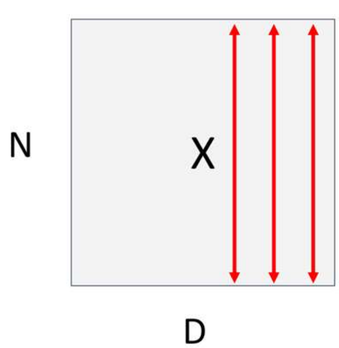


- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Not well-understood theoretically (yet)
- Behaves differently during training and testing: this is a very common source of bugs!

Batch Normalization

24

Input: $x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

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

25

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

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

Per-channel mean, shape is D

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

Per-channel std, 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

Prof. Monidipa Das, Department of CSE, IIT (ISM) Dhanbad

Batch Normalization

26

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

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

Per-channel mean, shape is D

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

Per-channel std, 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: Estimates depend on minibatch; can't do this at test-time!

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

Output, Shape is N x D

Prof. Monidipa Das, Department of CSE, IIT (ISM) Dhanbad

Batch Normalization

27

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

$\mu_j =$ (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 std, 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

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

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

Output, Shape is N x D

Batch Normalization for ConvNets

28

Batch Normalization for **fully-connected** networks

$$x : N \times D$$

Normalize

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

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

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

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

Normalize

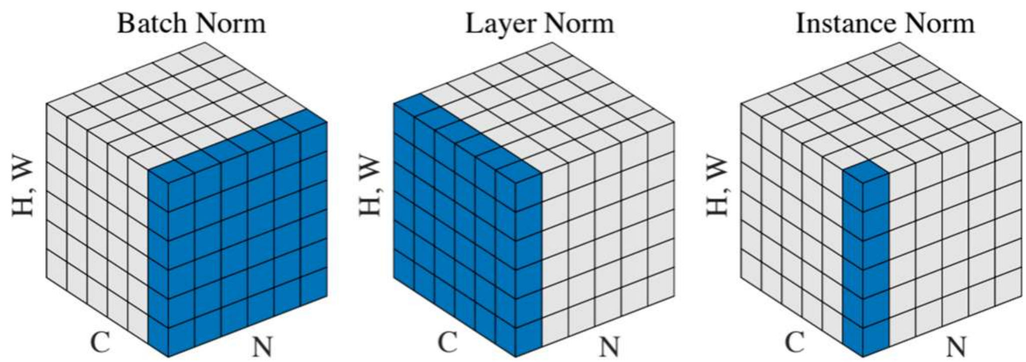
$$\mu, \sigma : 1 \times C \times 1 \times 1$$

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Comparison of Normalization Layers

29



Layer Normalization

30

Batch Normalization for **fully-connected** networks

$$x : N \times D$$

Normalize

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

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Layer Normalization for fully-connected networks
Same behavior at train and test!
Used in RNNs, Transformers

$$x : N \times D$$

Normalize

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

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Instance Normalization

31

Batch Normalization for convolutional networks

$$\boxed{\begin{array}{rl} & x : N \times C \times H \times W \\ \text{Normalize} & \quad\downarrow \qquad\qquad \downarrow \quad\; \downarrow \\ \mu, \sigma & : 1 \times C \times 1 \times 1 \end{array}}$$

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

$$y = \frac{(x - \mu)}{\sigma} \gamma + \beta$$

Instance Normalization for convolutional networks

$$\boxed{\begin{array}{rl} & x : N \times C \times H \times W \\ \text{Normalize} & \quad\downarrow \qquad\qquad \downarrow \quad\; \downarrow \\ \mu, \sigma & : N \times C \times 1 \times 1 \end{array}}$$

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

$$y = \frac{(x - \mu)}{\sigma} \gamma + \beta$$

32

Questions?

Acknowledgement: Prof. W. L. Hamilton, Fei-Fei Li, Ranjay Krishna, Danfei Xu, Justin Johnson, Xavier Bresson