Tutorial:
Best Practices of Convolutional Neural Networks

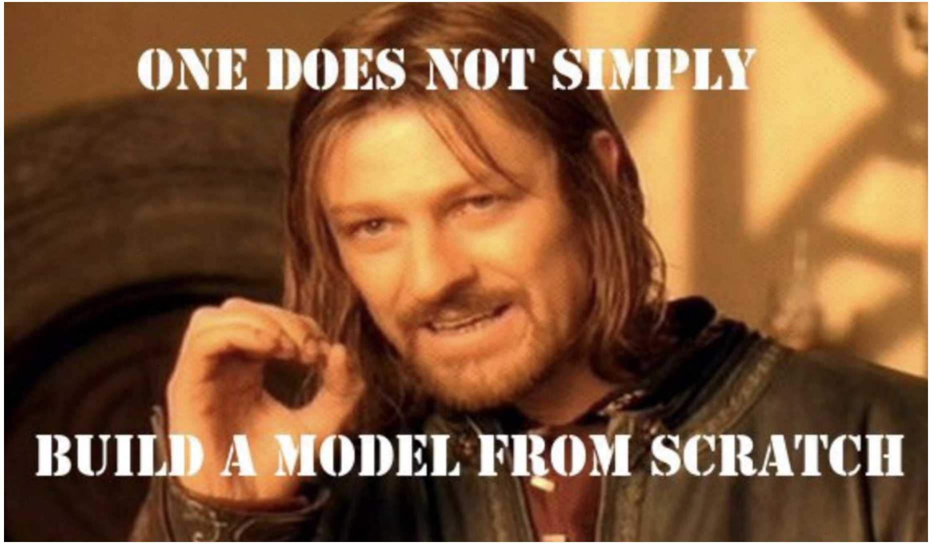
By Anusha Misra

am5684@columbia.edu

What we'll cover

- Transfer Learning
- Label Imbalance
- Normalization

Transfer Learning


ONE DOES NOT SIMPLY
BUILD A MODEL FROM SCRATCH

- What is it?

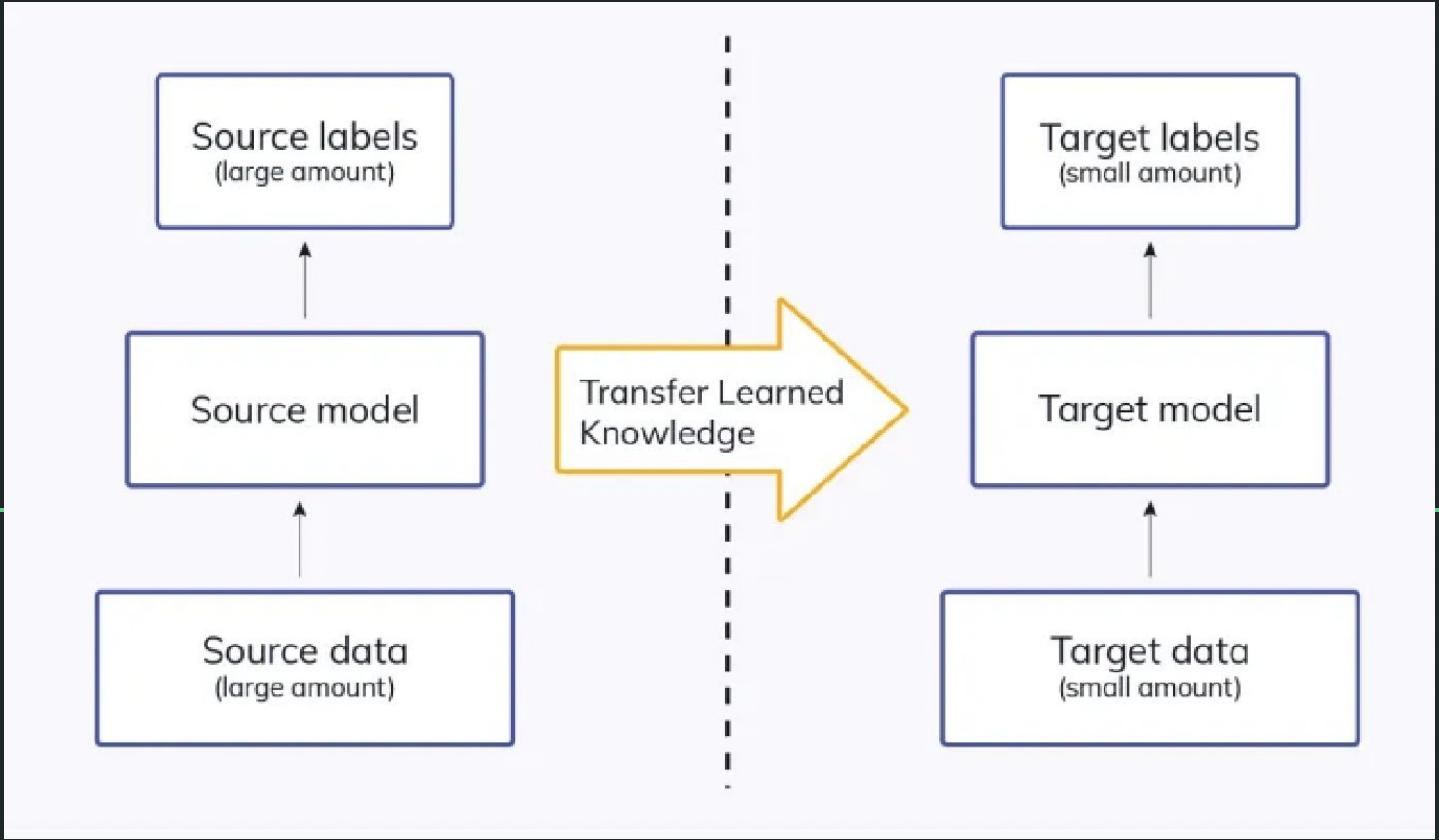
Pre-trained models are used as the starting point for a model on a second task.

- Why do we need it?

Small datasets may cause overfitting or bias.

- When is it useful?

This turns out to be very useful when the **training dataset** and the **computing power** both are limited. It helps improve **generalization** too.


Source labels
(large amount)
Source model
Source data
(large amount)
Transfer Learned
Knowledge
Target labels
(small amount)
Target model
Target data
(small amount)

Transfer Learning Scenarios

- $\mathbf{X_S \neq X_T}$: The feature spaces of the source and target domain are different.
- $\mathbf{P(X_S) \neq P(X_T)}$: The marginal probability distributions of source and target domain are different.
- $\boldsymbol{y_S \neq y_T}$: The label spaces between the two tasks are different.
- $\mathbf{P(\boldsymbol{y}_S|X_S) \neq P(\boldsymbol{y}_T|X_T)}$: The conditional probability distributions of the source and target tasks are different.

What’s transferred?

- **Instance**: Utilize training instances from the source domain for improvements in the target task.
- **Feature Representation**: Identifying good feature representations that can be utilized from the source to target domains.
- **Parameters**: Assumption that the models for related tasks share some parameters or prior distribution of hyperparameters.
- **Relational-Knowledge**: Deals with data, where each data point has a relationship with other data points.

How do you pick the Source Model?

In the source dataset, we get the feature representation for each input image. This is then averaged out to get 1 vector representing the source dataset.

This is then repeated for the target dataset.

Check for similarity between the feature representations of the source and target dataset.

What do you do with the layers of the Source Model?

Freeze or Fine-Tune?

The bottom *n* layers can be frozen or fine-tuned.

Learning rates can be played around with to find a trade-off between freezing a layer or fine tuning it.

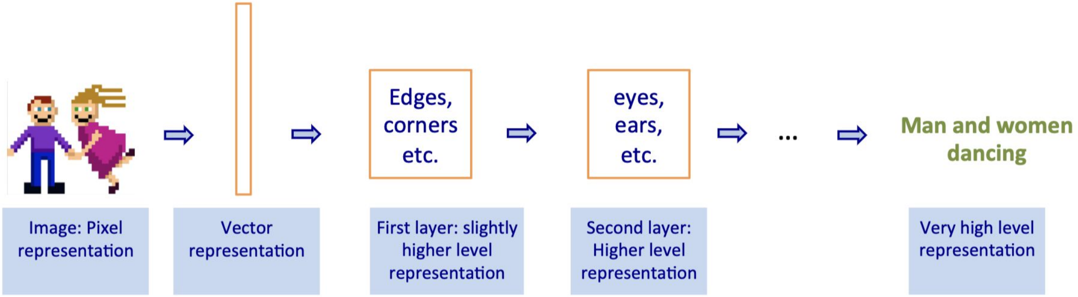
Why not use the same learning rate?

Same can’t be used because it caters to different kind of data and model.
Smaller learning rates: Fine-tunes better but might take longer.
Larger learning rates can deviate the model from its learnt weights, thus making it forget what it previously learnt.

Why does Fine-tuning work?

The initial layers learn generic information - not target specific. These can be frozen since this generic information is still needed and in the same capacity.
Deeper layers are more problem/target specific. They can be fine-tuned to fit the particular needs of the problem at hand.

How the Network Works



Transfer Learning: Rule of Thumb

	Target Dataset is small	Target Dataset is large
Similar to Source dataset	**Freeze**	**Fine-tune many**
Dissimilar to Source dataset	**Train SVM from low-level features first**	**Train from scratch**

Transfer Learning from ImageNet

When dealing with a classification model, a good choice of a random source dataset to begin with is **ImageNet**.

ImageNet properties:

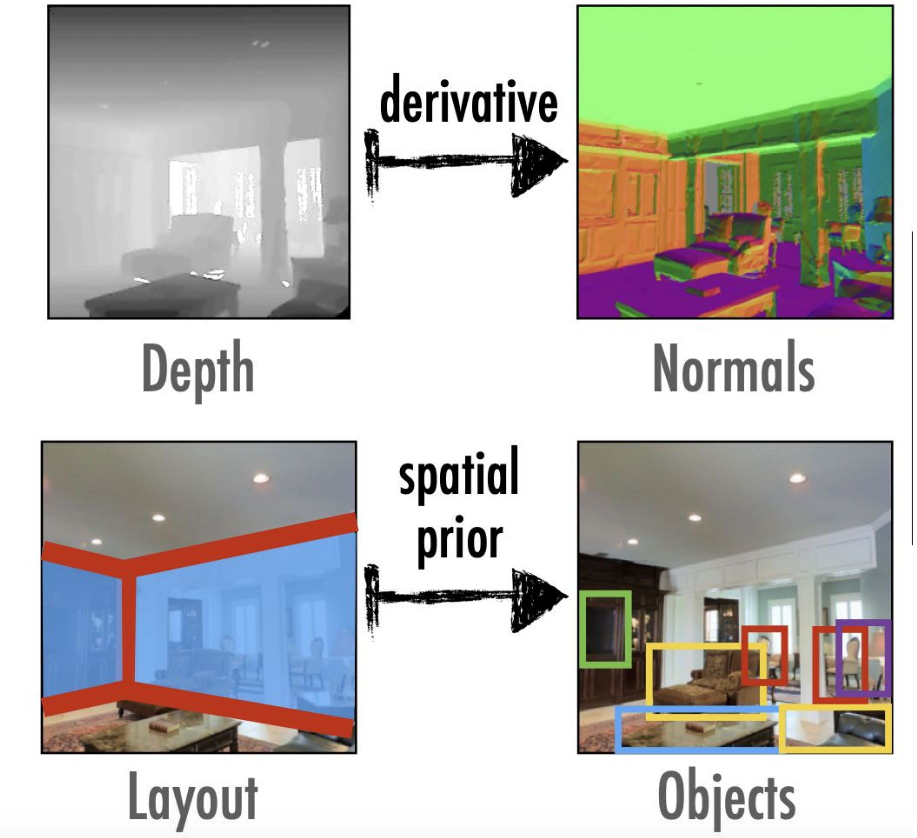
- 130M images
- Hierarchical labels (eg: Vehicle: Planes, etc)
- Can be used for object detection
- More than 20k categories (labels)

(for more details and to participate - ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Task Transfer Learning

- Same domain, different tasks
- Task relationships exist
- Can be computationally measured
- Tasks belonging to a structured space

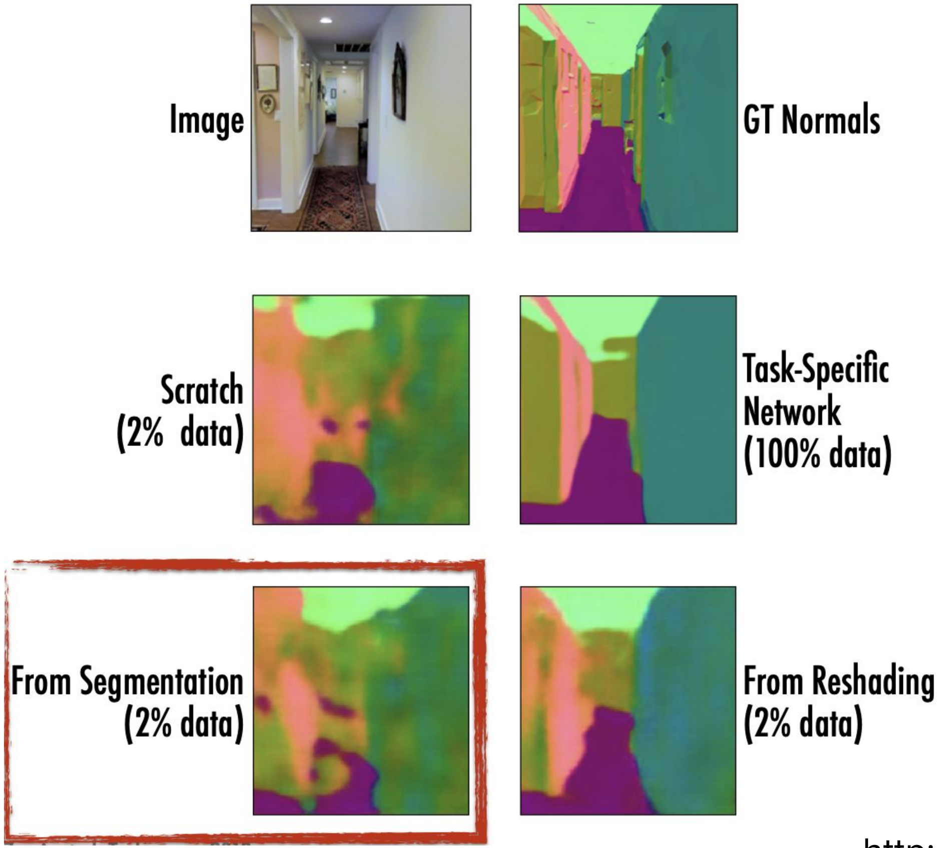
http://taskonomy.stanford.edu/taskonomy_CVPR2018.pdf

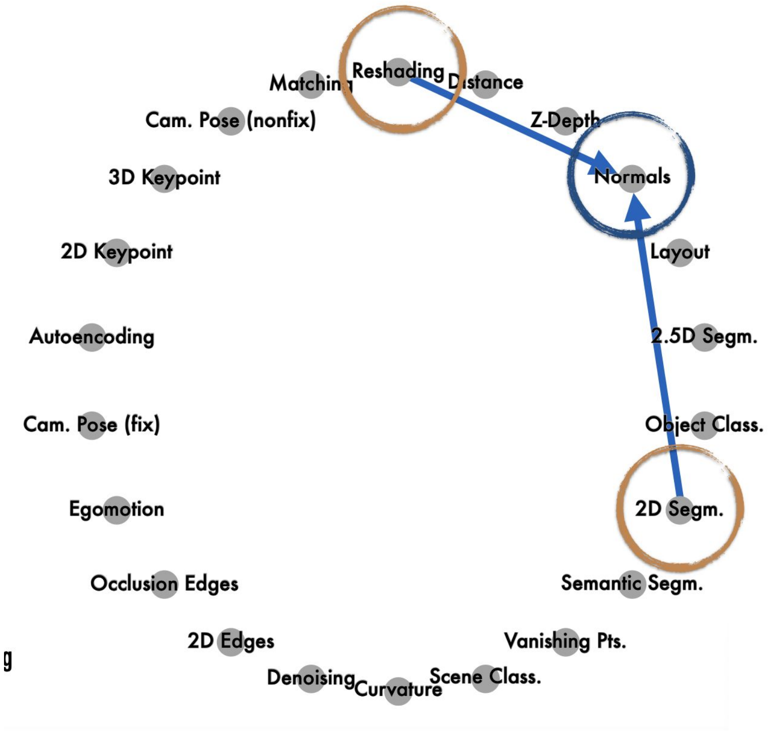


Image

http://taskonomy.stanford.edu/taskonomy_CVPR2018.pdf

Task Transfer Learning Examples





http://taskonomy.stanford.edu/taskonomy_CVPR2018.pdf

Disadvantages of Transfer Learning

- Negative transfer
- It's very hard to get it right but very easy to mess up!

Label Imbalance

What is it?

Data where the classes are not represented equally. **Why is this bad?**

Model doesn’t have enough data to learn relationship between features and each class properly.

Example: Detection of cancer or anomaly detection in general, spam filtering

How do we fix this?

- Resampling the dataset
  - Adding copies of instances from the under-represented class called over-sampling
  - Deleting instances from the over-represented class, called under-sampling.
- Reweight the loss by class ratio
- Batch Sampling

Choosing a Performance Metric for Label Imbalance

Positives more than Negatives

- FPR: is high. Since our model predicts everything 1, we have a high number of FP. And it signifies that this is not a good classifier/model.
- AUC score: is very low and represents the true picture of evaluation here.

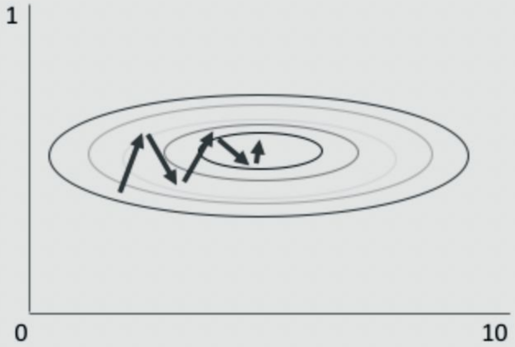
Choosing a Performance Metric for Label Imbalance

Negatives more than Positives

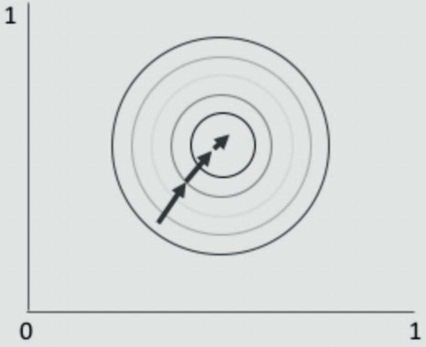
- Precision: is very low. Because of the high number of FP
  The ration of TP/(TP+FP) becomes low.
- Recall: is very low. Since data has a very disproportionately high number of Negative cases. The classifier may detect a larger no. of positive as negative.
  The ration of TP/(TP+FN) becomes low.
- F1-score: is low. The low values of Precision and Recall make F1-score, a good indicator of performance here.

Normalization

Why normalize?



Gradient of larger parameter
dominates the update



Both parameters can be
updated in equal proportions

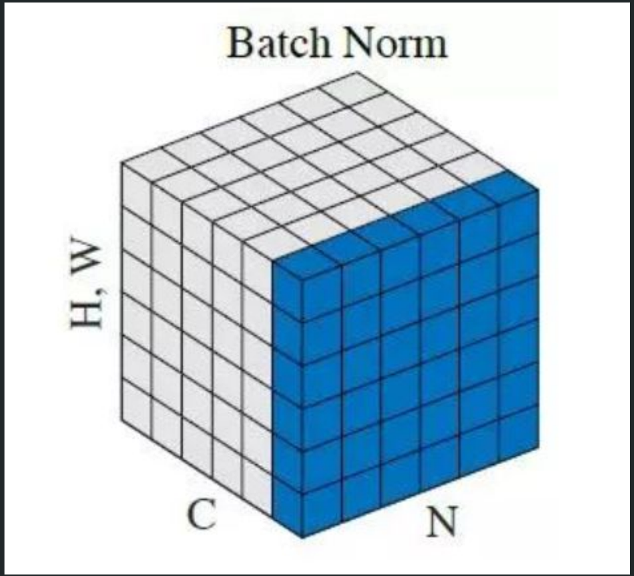
Types of Normalization

- Batch normalization
- Layer normalization
- Instance normalization
- Group normalization

Batch Normalization

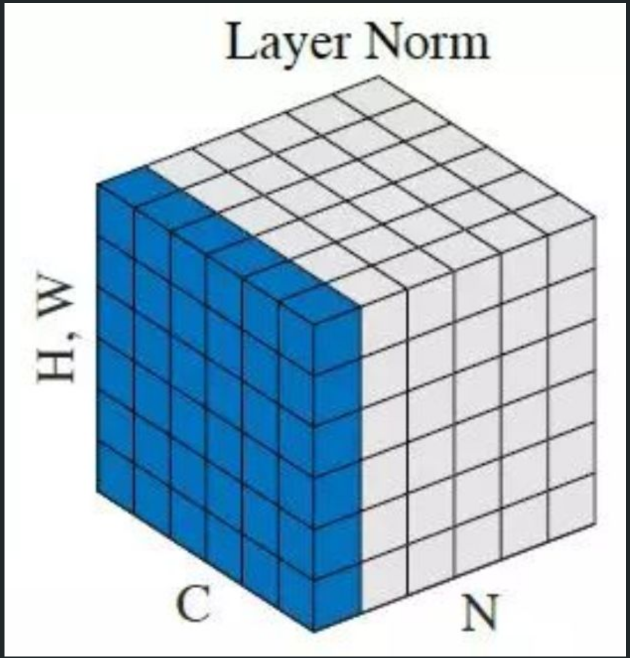
Scales the inputs to a layer to a common value for every mini-batch during the training of deep neural networks.

The network trains faster!



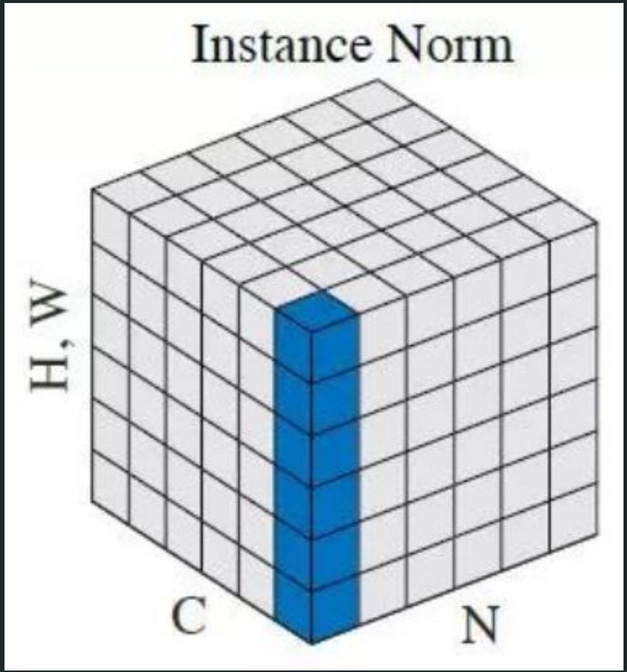
Layer Normalization

Normalizes the summed input across the features.



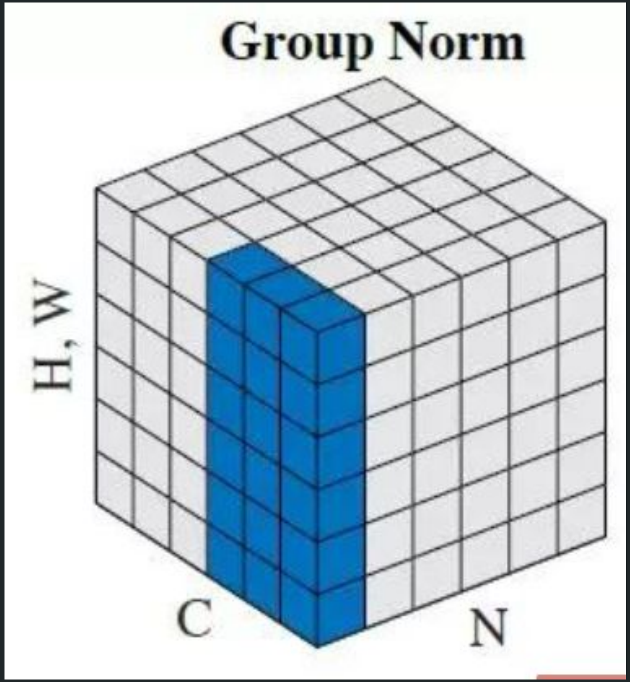
Instance Normalization

Normalizes across each channel of the training data



Group Normalization

Divides the channels into groups and normalizes them for each training example



Sync Batch Norm

- Split large batch into several and distribute them many GPUs
- Collect the batch statistics from all devices

Batch Normalization Disadvantages

- With a batch size of 1, the variance would be 0 (x_{norm}=x), which defeats the purpose, and batch normalization wouldn't work.

$$x_{normalized} = \frac{x - m}{s}$$

- Increased training time, extra computation
- Different results for training and test

Thank you!