Tutorial:
(Some) Best Practices of ConvNet Application

Sheng Jia

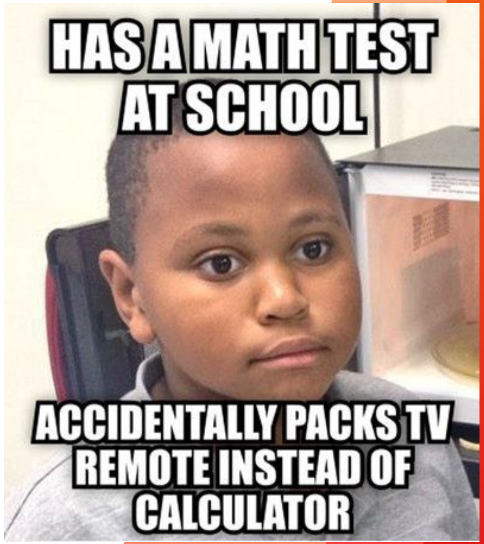
Feb. 15, 2022

(Adapted from Jenny Bao's slides in winter 2021)

“

~~Math-heavy~~ tutorial

-> High-level guidance



Overview

- Transfer Learning
- Label Imbalance
- Normalization

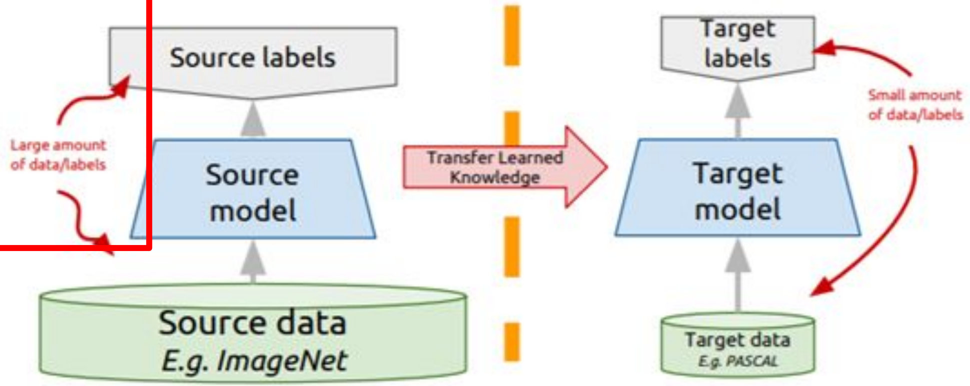
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



Freeze or fine-tune?

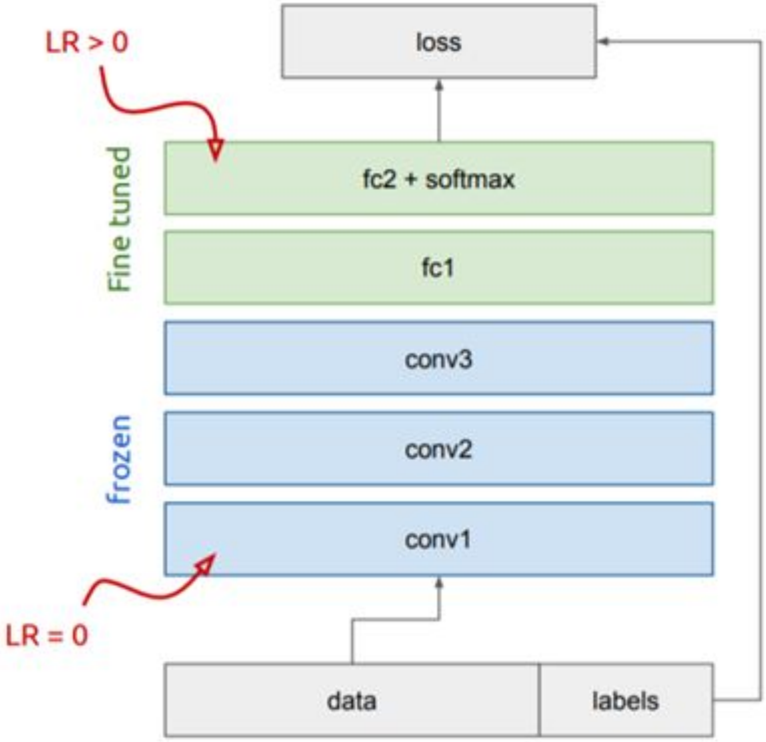
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



Transfer Learning: Rule of thumb

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

http://cs231n.github.io/transfer-learning/#tf

Transfer Learning

- Additional advice:
  - Smaller learning rate when fine-tuning

http://cs231n.github.io/transfer-learning/#tf

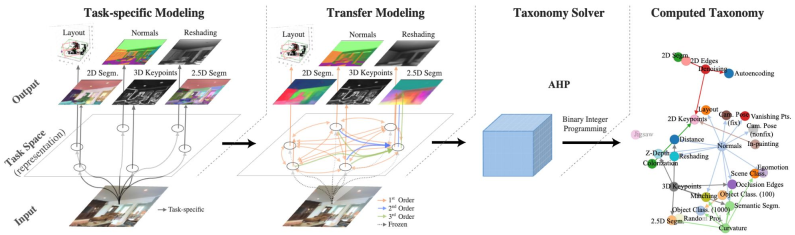
Task Transfer Learning

- Same domain, different tasks
- Computer Vision Taskonomy:http://taskonomy.stanford.edu
- What is the relation between *3d keypoint detection* and *depth estimation?*

Taskonomy: Disentangling Task Transfer Learning, Amir et al, 2018

Task Transfer Learning

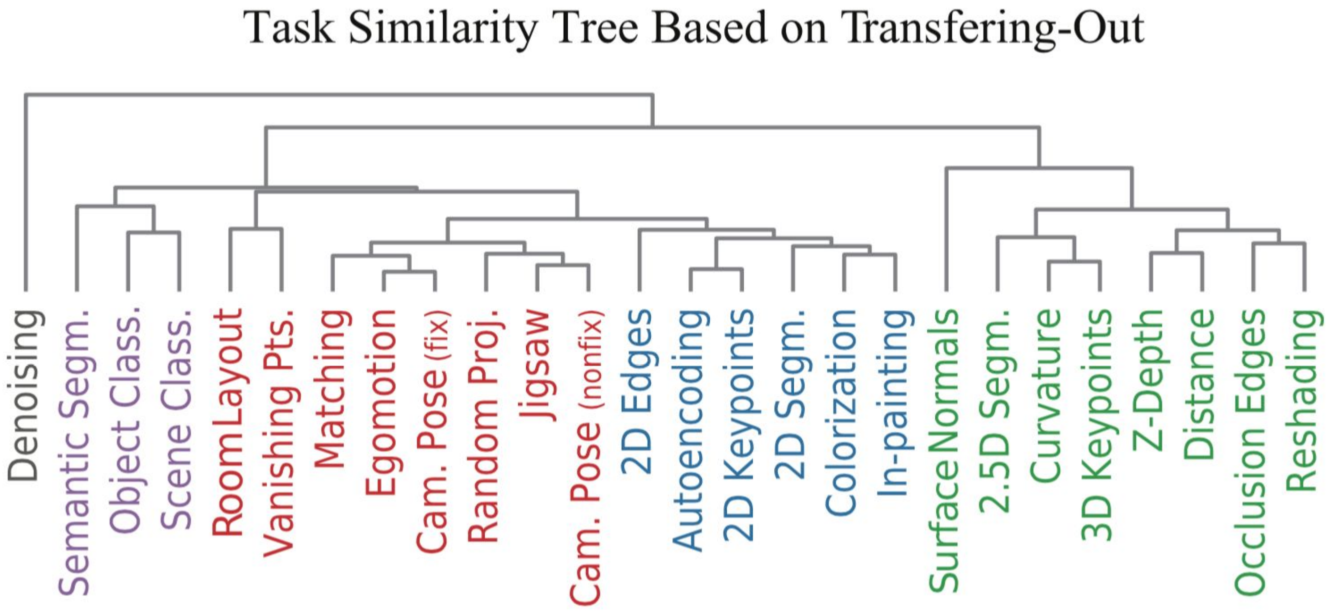
- Same domain, different tasks
- Computer Vision Taskonomy: http://taskonomy.stanford.edu
- What is the relation between *3d keypoint detection* and *depth estimation*?
- Is it able to structurally represent them?



Process overview. The steps involved in creating the taxonomy.

Taskonomy: Disentangling Task Transfer Learning, Amir et al, 2018

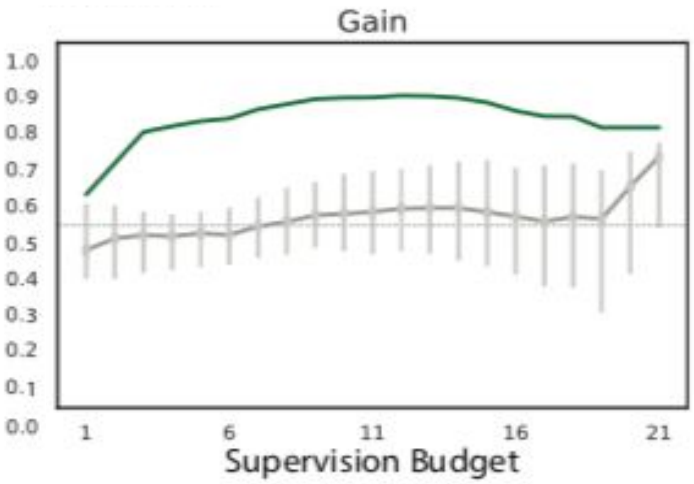
Task Transfer Learning



[Taskonomy: Disentangling Task Transfer Learning](), Amir et al, 2018

Task Transfer Learning: Result

- How significant is the discovered structure of task space?



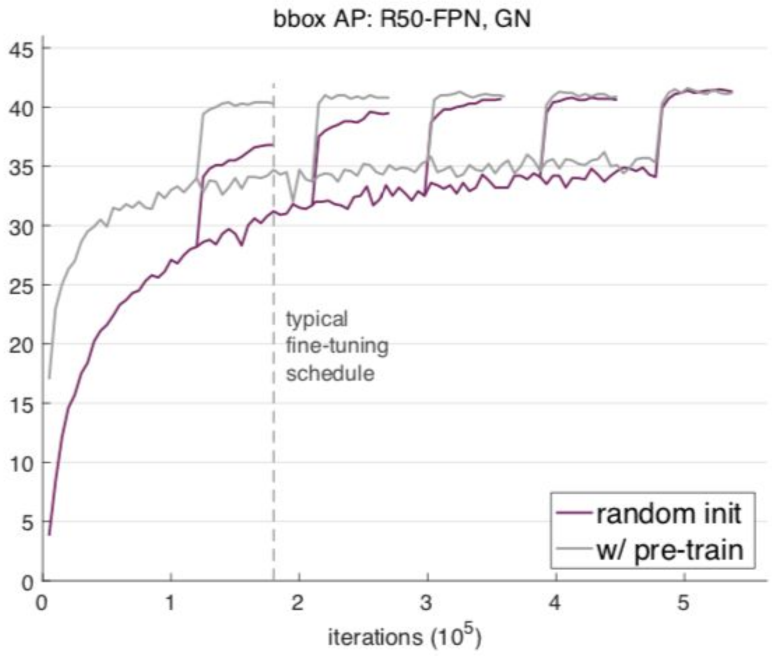
Green: look up the taxonomy connectivities.

Gray: use random connectivities

Taskonomy: Disentangling Task Transfer Learning, Amir et al, 2018

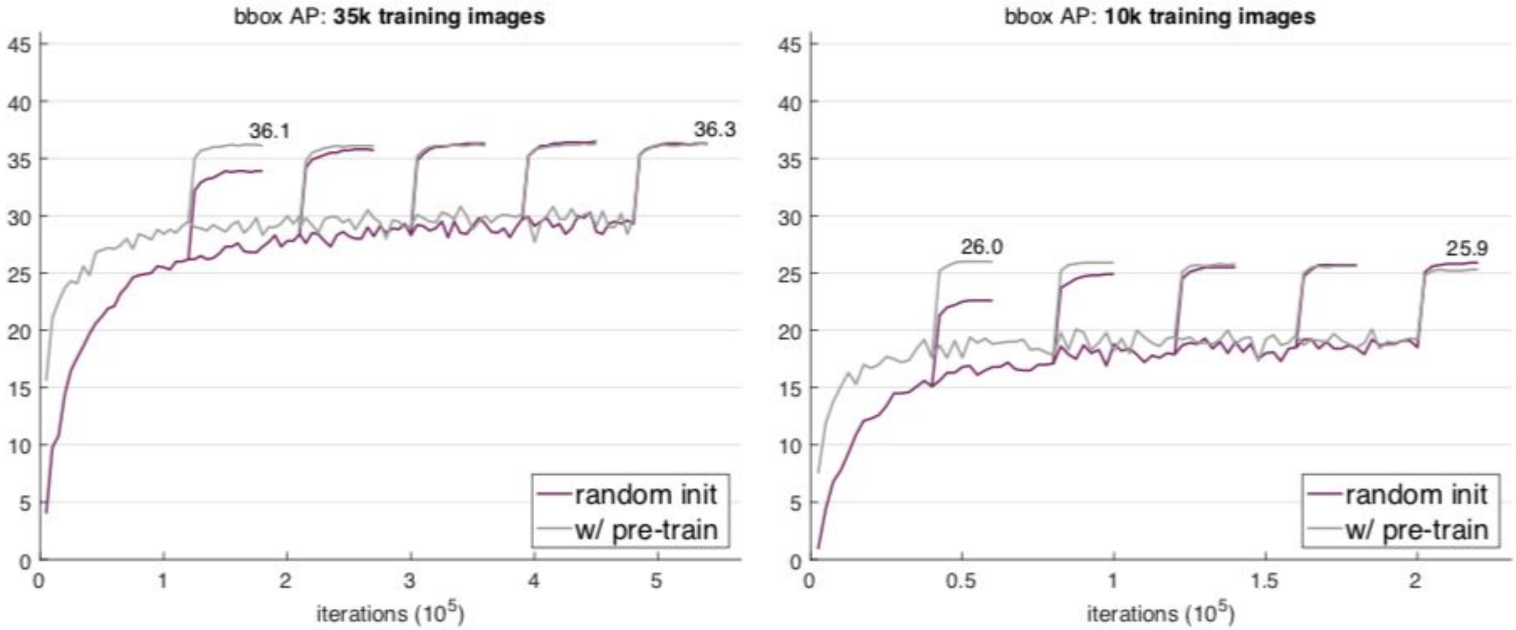
Transfer Learning from ImageNet?

- Always better?
- ImageNet: 130M
- COCO: 8.6M



Rethinking ImageNet Pre-training, Kaiming et al, 2019

Transfer Learning from ImageNet?



Rethinking ImageNet Pre-training, Kaiming et al, 2019

Transfer Learning from ImageNet?

- With only 1k training image:
  - w/ pretrain: 9.9 AP
  - Random init: 3.5 AP

  (on validation set)

Overfitting without transfer learning

Train loss similar at convergence but validation error different this time!

Rethinking ImageNet Pre-training, Kaiming et al, 2019

Transfer Learning from ImageNet?

One conclusion:

Training from scratch **can be no worse** than its ImageNet pre-training counterparts under many circumstances, **down to** 10k COCO images.

Rethinking ImageNet Pre-training, Kaiming et al, 2019

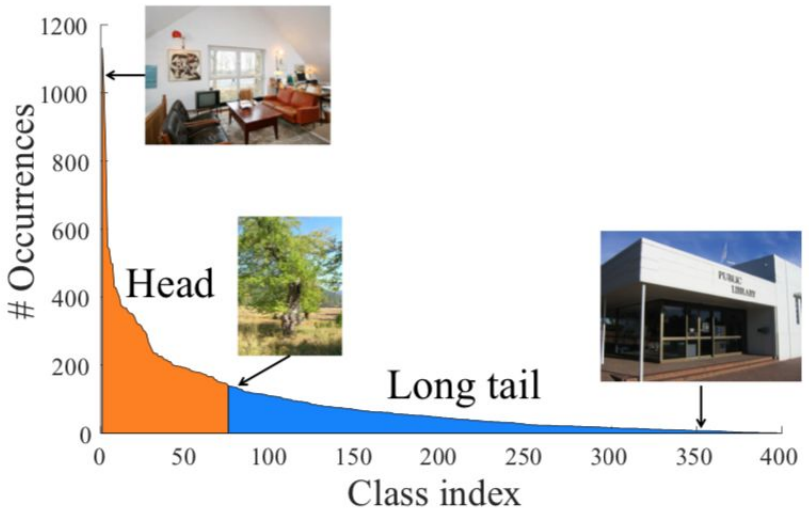
Transfer Learning: Rule of thumb

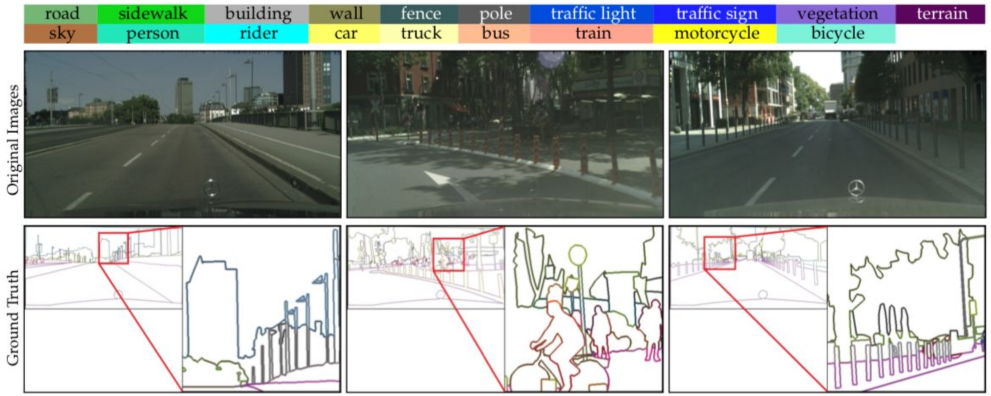
	|Target Dataset| is small	|Target Dataset| is large
Similar to Source dataset	**Freeze**	**Fine-tune all**
Dissimilar to Source dataset	**Try SVM from low-level features first**	**Train from scratch**

http://cs231n.github.io/transfer-learning/#tf

Label Imbalance

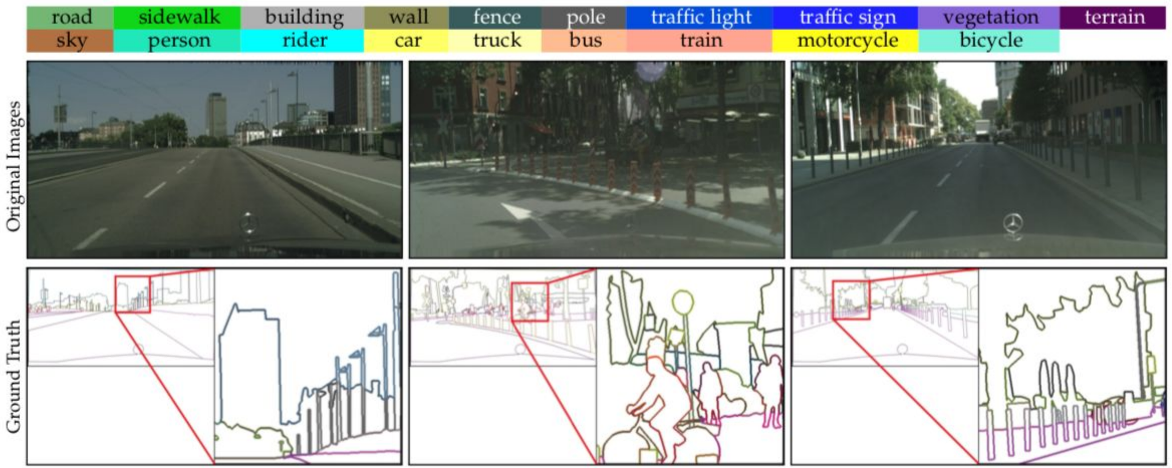
- Semantic Segmentation
- Contour Detection
- Long tail recognition





Label Imbalance

- Reweight the loss by class ratio
- Data Resampling by class ratio



Structure of ConvNet

- Conv -> **Normalization** -> ReLU -> Pooling

Normalization layers

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \boxed{\gamma} + \boxed{\beta}$$

Learnable parameters, to make sure the normalization layer can represent identity transformation

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

BatchNorm

- Internal Covariate Shift
- Compute batch statistic during training
  - Dependent on mini-batch

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

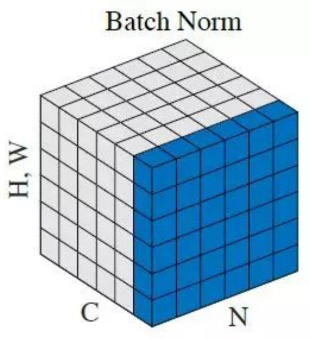
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

BatchNorm for CNN

- Jointly normalize all the activations in a minibatch, over all locations.
  - "Effective minibatch size of N'=N*H*W.
  - Learn a pair of parameters per feature map, rather than per activation.



BatchNorm

- Usually, during training, BN keeps a running estimate of the mean and variance, which are used at testing time.

  Recall:

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Running estimates of E[x], Var[x] besides learnable parameters.

BatchNorm Example

Pytorch documentation

CLASS `torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)` [SOURCE]

- num_features: C from an expected input of size (N, C, H, W)

Example: convolution block in Inception Net V3

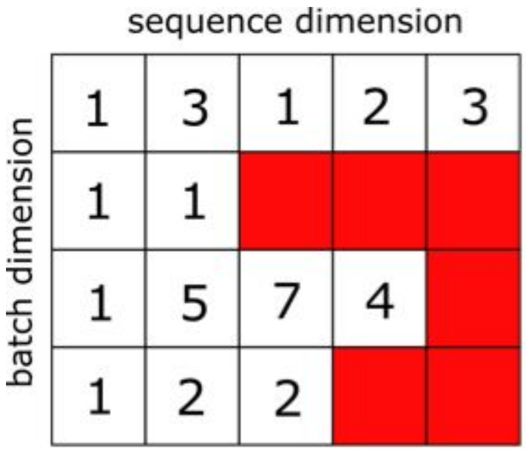
```
class BasicConv2d(nn.Module):

    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        **kwargs: Any
    ) -> None:
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
        self.bn = nn.BatchNorm2d(out_channels, eps=0.001)

    def forward(self, x: Tensor) -> Tensor:
        x = self.conv(x)
        x = self.bn(x)
        return F.relu(x, inplace=True)
```

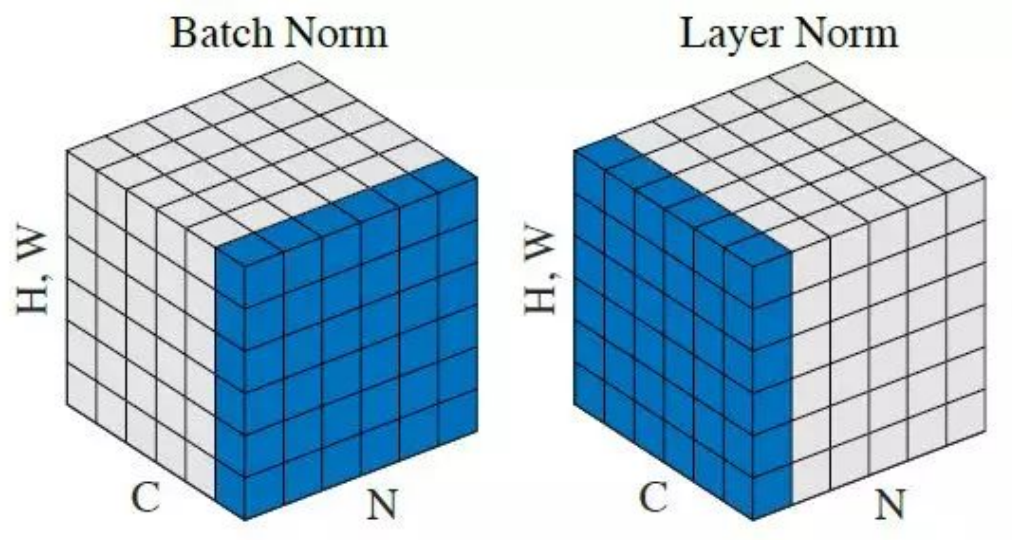
BatchNorm -- limitations

- Performance depends on the batch size
- Difficult to apply to recurrent connections



LayerNorm

- Normalize across the entire layer for each training example.



LayerNorm

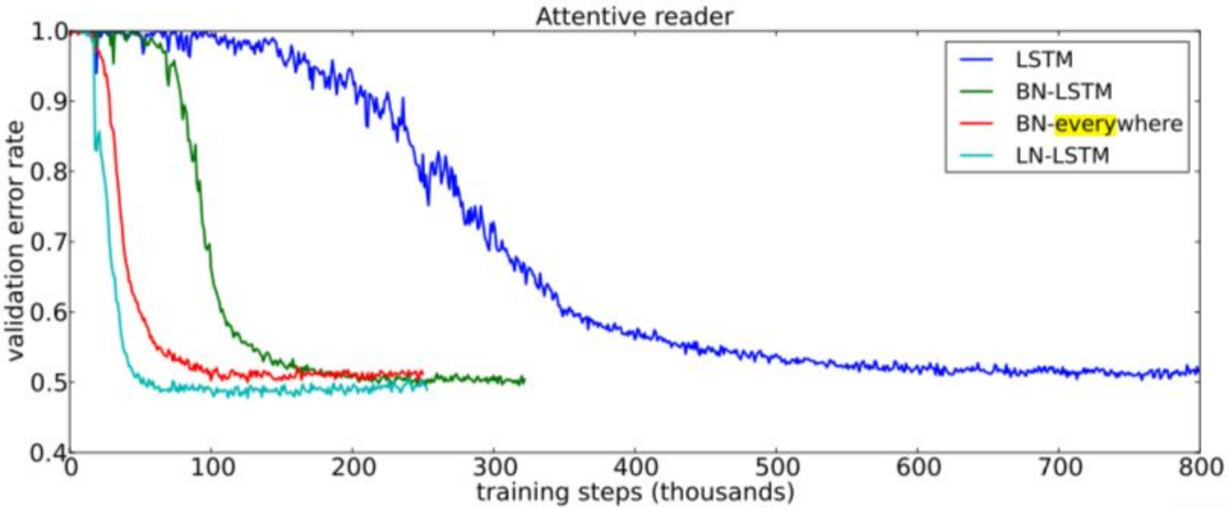
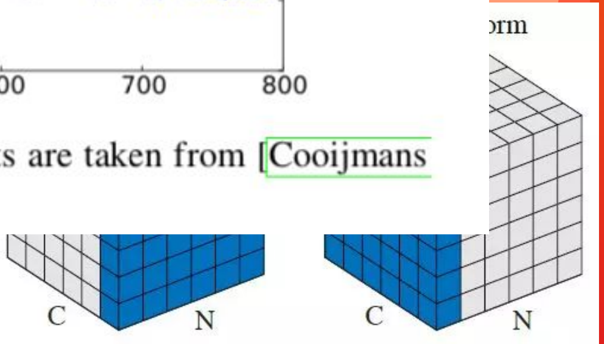


Figure 2: Validation curves for the attentive reader model. BN results are taken from [Cooijmans et al., 2016].



LayerNorm Example

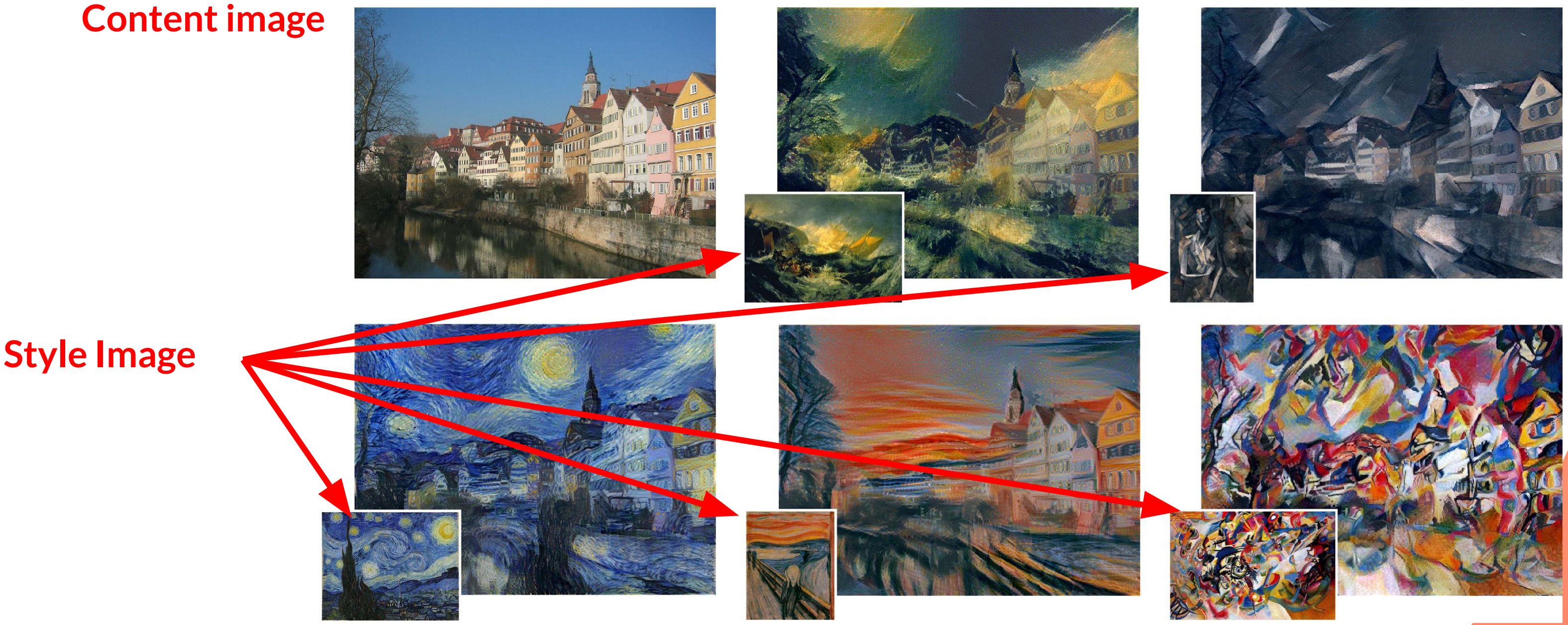
[Pytorch documentation]

```
CLASS torch.nn.LayerNorm(normalized_shape: Union[int, List[int],
      torch.Size], eps: float = 1e-05, elementwise_affine: bool = True)    [SOURCE]
```

```
input = torch.randn(20, 5, 10, 10)
# With Learnable Parameters
m = nn.LayerNorm(input.size()[1:])
# Without Learnable Parameters
m = nn.LayerNorm(input.size()[1:], elementwise_affine=False)
# Normalize over last two dimensions
m = nn.LayerNorm([10, 10])
# Normalize over last dimension of size 10
m = nn.LayerNorm(10)
# Activating the module
output = m(input)
```

InstanceNorm

- Special Case: Feed-Forward Stylization



InstanceNorm

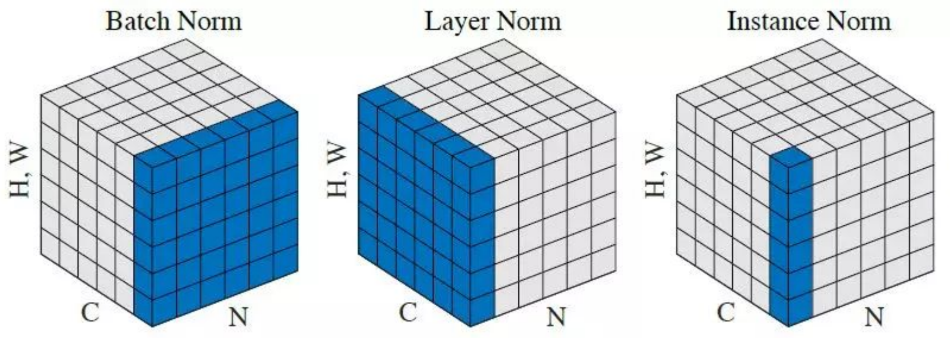
- Special Case: Feed-Forward Stylization
- Invariant to the **contrast** (style) of the content image

InstanceNorm

- Special Case: Feed-Forward Stylization
- Invariant to the **contrast** (style) of the content image
- **Channel-wise** normalization

InstanceNorm

- Special Case: Feed-Forward Stylization
- Invariant to the **contrast** of the content image
- Normalize over channel for each image



InstanceNorm Example

[Pytorch documentation]()

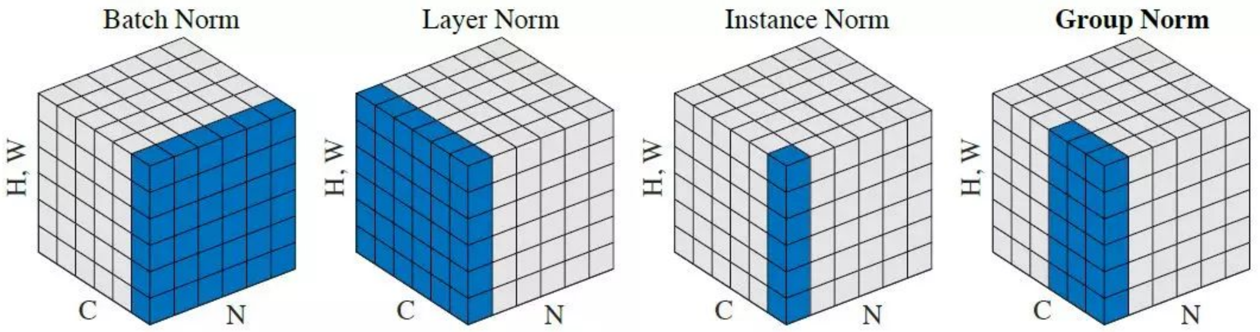
CLASS torch.nn.InstanceNorm2d(*num_features: int, eps: float = 1e-05, momentum: float = 0.1, affine: bool = False, track_running_stats: bool = False*) [SOURCE]

- num_features: C from an expected input of size (N, C, H, W)
- By default, there are no learnable parameters, and does not track running statistics (unlike BN or LN)

```
# Without Learnable Parameters
m = nn.InstanceNorm2d(100)
# With Learnable Parameters
m = nn.InstanceNorm2d(100, affine=True)
input = torch.randn(20, 100, 35, 45)
output = m(input)
```

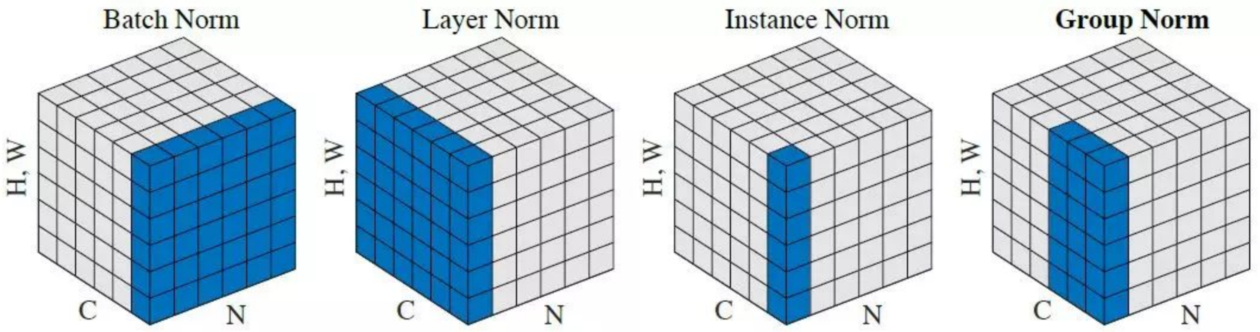
GroupNorm

- Large Feed-Forward network
  - Sometimes batch size is small due to computational constraints
- How to adjust?
  - GroupNorm

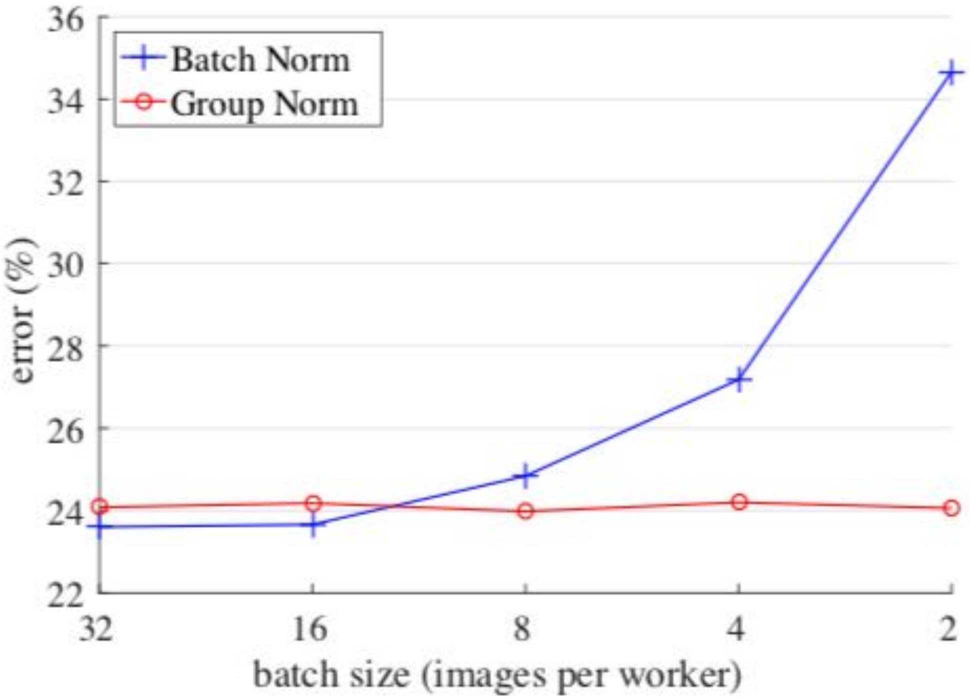


GroupNorm

- Group a set of features and normalize them
  - like normalizing HOG and SIFT separately



GroupNorm



Batch Norm
Group Norm
error (%)
36
34
32
30
28
26
24
22
32
16
8
4
2
batch size (images per worker)

GroupNorm Example

Pytorch documentation

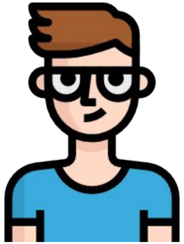
CLASS `torch.nn.GroupNorm(num_groups: int, num_channels: int, eps: float = 1e-05, affine: bool = True)` [SOURCE]

- num_groups (*int*) – number of groups to separate the channels into
- num_channels (*int*) – number of channels expected in input

```
input = torch.randn(20, 6, 10, 10)
# Separate 6 channels into 3 groups
m = nn.GroupNorm(3, 6)
# Separate 6 channels into 6 groups (equivalent with InstanceNorm)
m = nn.GroupNorm(6, 6)
# Put all 6 channels into a single group (equivalent with LayerNorm)
m = nn.GroupNorm(1, 6)
# Activating the module
output = m(input)
```

SyncBatchNorm

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



Any Question?