Tutorial: (Some) Best Practices of ConvNet Application

Jenny Bao
Feb. 25, 2021
(adapted from Andrew Liao’s slides in winter 2020)
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

<table>
<thead>
<tr>
<th>Target Dataset</th>
<th>is small</th>
<th>Target Dataset</th>
<th>is large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar to Source dataset</td>
<td>Freeze</td>
<td>Fine-tune all</td>
<td></td>
</tr>
<tr>
<td>Dissimilar to Source dataset</td>
<td>Try SVM from low-level features first</td>
<td>Train from scratch</td>
<td></td>
</tr>
</tbody>
</table>
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](http://taskonomy.stanford.edu)
- What is the relation between 3d keypoint detection and depth estimation?
Task Transfer Learning

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

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

Task Similarity Tree Based on Transferring-Out

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

- How significant is the discovered structure of task space?

[Diagram showing the relationship between supervision budget and gain]
Transfer Learning from ImageNet?

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

Rethinking ImageNet Pre-training, Kaiming et al, 2019
Transfer Learning from ImageNet?

- Always better?

**ImageNet: 130M**

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

Rethinking ImageNet Pre-training, Kaiming et al, 2019
## Transfer Learning: Rule of thumb

<table>
<thead>
<tr>
<th></th>
<th>Target Dataset is small</th>
<th>Target Dataset is large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar to Source dataset</td>
<td>Freeze</td>
<td>Fine-tune all</td>
</tr>
<tr>
<td>Dissimilar to Source dataset</td>
<td>Try SVM from low-level features first</td>
<td>Train from scratch</td>
</tr>
</tbody>
</table>
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

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

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
BatchNorm

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

Pytorch documentation

```python
class torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

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

Example: convolution block in Inception Net V3

```python
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

```python
input = torch.randn(20, 5, 10, 10)
# With Learnable Parameters
m = nn.LayerNorm(input.size()[1:]), elementwise_affine=False)
# 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**

- 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)

```python
# 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
GroupNorm Example

Pytorch documentation

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

- 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?