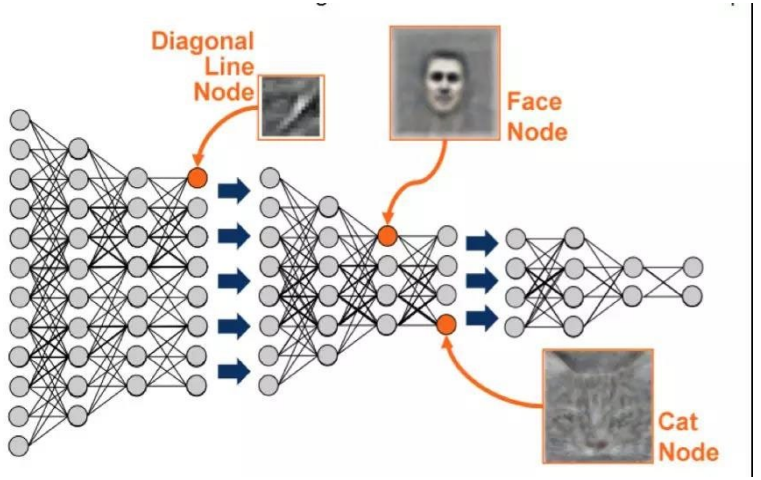
Training model on Pytorch

吴斌

1. Fine-tune

我们在完成自己的task的时候，往往需要训练一个泛化性好的模型。这时需要大量的训练数据。数据有限？

使用large-scale数据集的预训练(pre-trained)参数迁移到模型，然后在我们自己数据集上微调(fine-tune)。我们称之为fine-tune。



1. Fine-tune

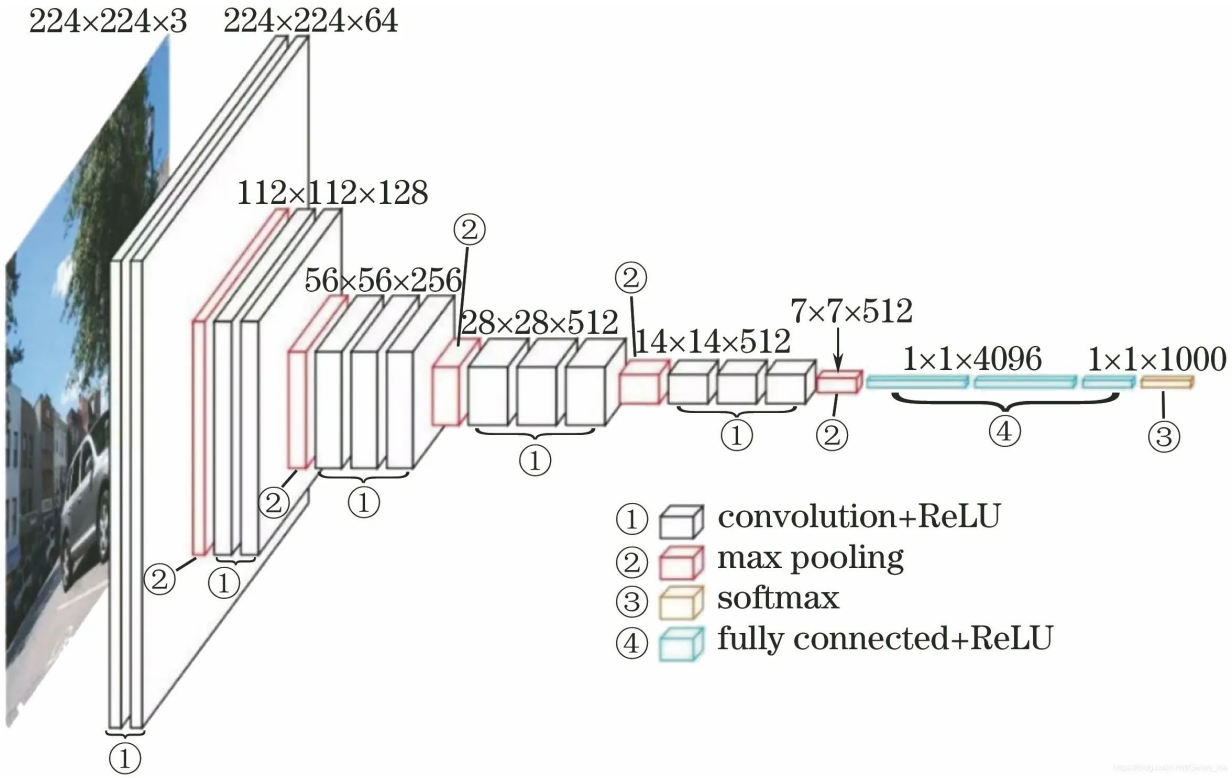
不收敛，泛化能力低

1. Model trained from scratch

2.Fully finetune

3. Finetune only the fully connected layer

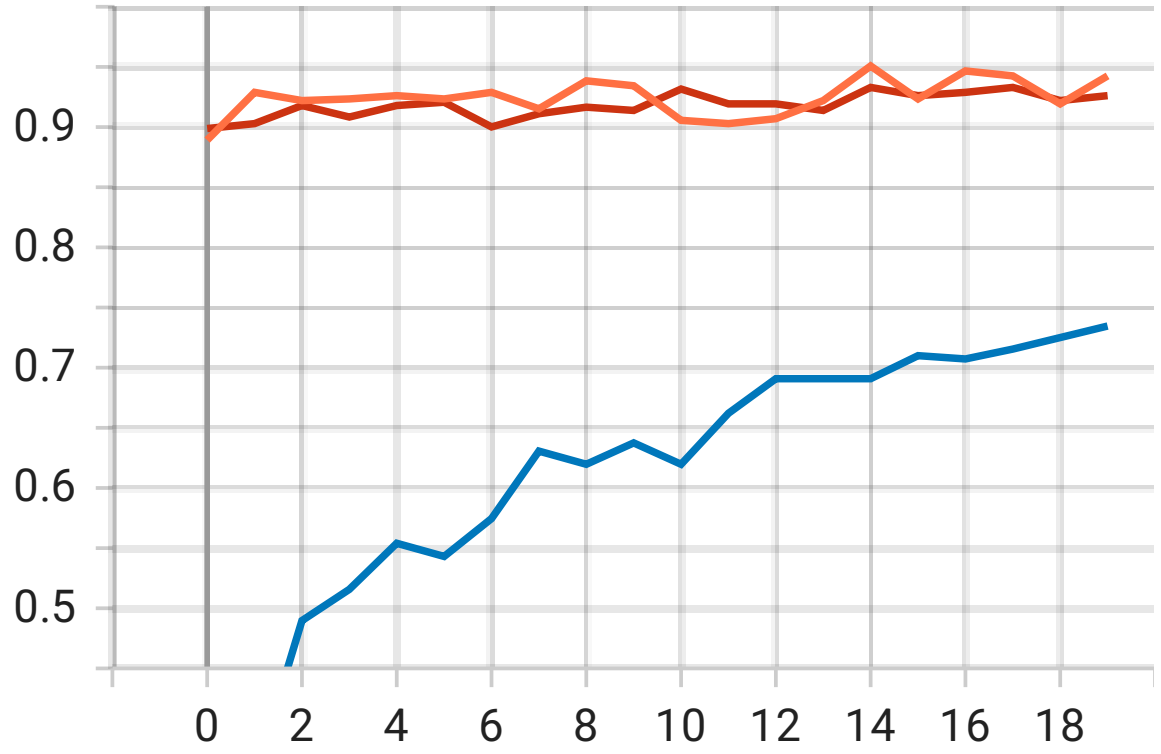
Freezing weights



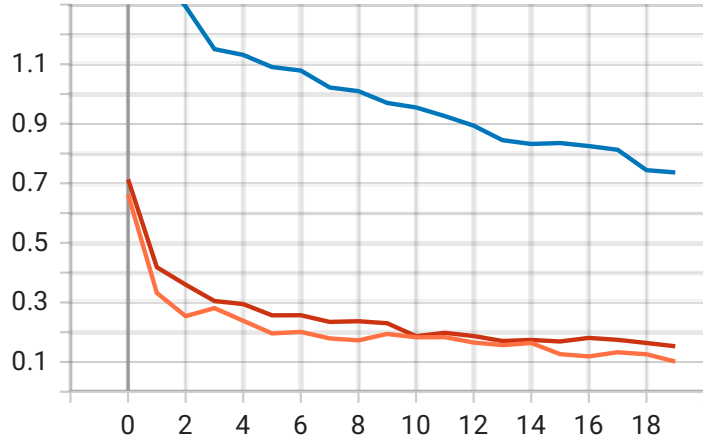
1. Fine-tune

Name	Smoothed	Value	Step
finetune	0.9425	0.9425	19
finetune-fc	0.9261	0.9261	19
scratch	0.7346	0.7346	19

val_acc



train_loss



1. Fine-tune

权值文件.pth

类似Python字典的键值对存储权值

```python
model = vgg(model_name=model_name, num_classes=5, init_weights=True)
model.to(device)
# for k, v in model.state_dict().items():
#     print(k)
pretrained_state = torch.load('/Pth/vgg16-397923af.pth', map_location='cpu')
state_dict = model.state_dict()
for k in ['classifier.6.weight', 'classifier.6.bias']:
    if k in pretrained_state and pretrained_state[k].shape != state_dict[k].shape:
        print(f"Removing key {k} from pretrained checkpoint")
        del pretrained_state[k]
model.load_state_dict(pretrained_state, strict=False)
```

```
features.0.weight shape:torch.Size([64, 3, 3, 3])
features.0.bias shape:torch.Size([64])
features.2.weight shape:torch.Size([64, 64, 3, 3])
features.2.bias shape:torch.Size([64])
features.5.weight shape:torch.Size([128, 64, 3, 3])
features.5.bias shape:torch.Size([128])
features.7.weight shape:torch.Size([128, 128, 3, 3])
features.7.bias shape:torch.Size([128])
features.10.weight shape:torch.Size([256, 128, 3, 3])
features.10.bias shape:torch.Size([256])
features.12.weight shape:torch.Size([256, 256, 3, 3])
features.12.bias shape:torch.Size([256])
features.14.weight shape:torch.Size([256, 256, 3, 3])
features.14.bias shape:torch.Size([256])
features.17.weight shape:torch.Size([512, 256, 3, 3])
features.17.bias shape:torch.Size([512])
features.19.weight shape:torch.Size([512, 512, 3, 3])
features.19.bias shape:torch.Size([512])
features.21.weight shape:torch.Size([512, 512, 3, 3])
features.21.bias shape:torch.Size([512])
features.24.weight shape:torch.Size([512, 512, 3, 3])
features.24.bias shape:torch.Size([512])
features.26.weight shape:torch.Size([512, 512, 3, 3])
features.26.bias shape:torch.Size([512])
features.28.weight shape:torch.Size([512, 512, 3, 3])
features.28.bias shape:torch.Size([512])
classifier.0.weight shape:torch.Size([4096, 25088])
classifier.0.bias shape:torch.Size([4096])
classifier.3.weight shape:torch.Size([4096, 4096])
classifier.3.bias shape:torch.Size([4096])
classifier.6.weight shape:torch.Size([5, 4096])
classifier.6.bias shape:torch.Size([5])
```

1. Fine-tune

如何冻结参数?

model.py

```
def forward(self, x):
    with torch.no_grad():
        # N x 3 x 224 x 224
        x = self.features(x)
    # N x 512 x 7 x 7
    x = torch.flatten(x, start_dim=1)
    # N x 512*7*7
    x = self.classifier(x)
    return x
```

注意事项：微调过程学习率降低，例如降低10倍。
0.001->0.0001

```
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

2. How to process different inputs?

(1)、通常我们不修改网络，尽量让resize图片的尺寸让其适应网络的输入。例如256->224。或者选择不同型号的模型，如Visual Transformer一般会提供224和384两种输入的网络。

256

256

Input=224x224

```
data_transform = {
    "train": transforms.Compose([transforms.RandomResizedCrop(224),
                                 transforms.RandomHorizontalFlip(),
                                 transforms.ToTensor(),
                                 transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])]),
    "val": transforms.Compose([transforms.Resize(256),
                               transforms.CenterCrop(224),
                               transforms.ToTensor(),
                               transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])])}
```

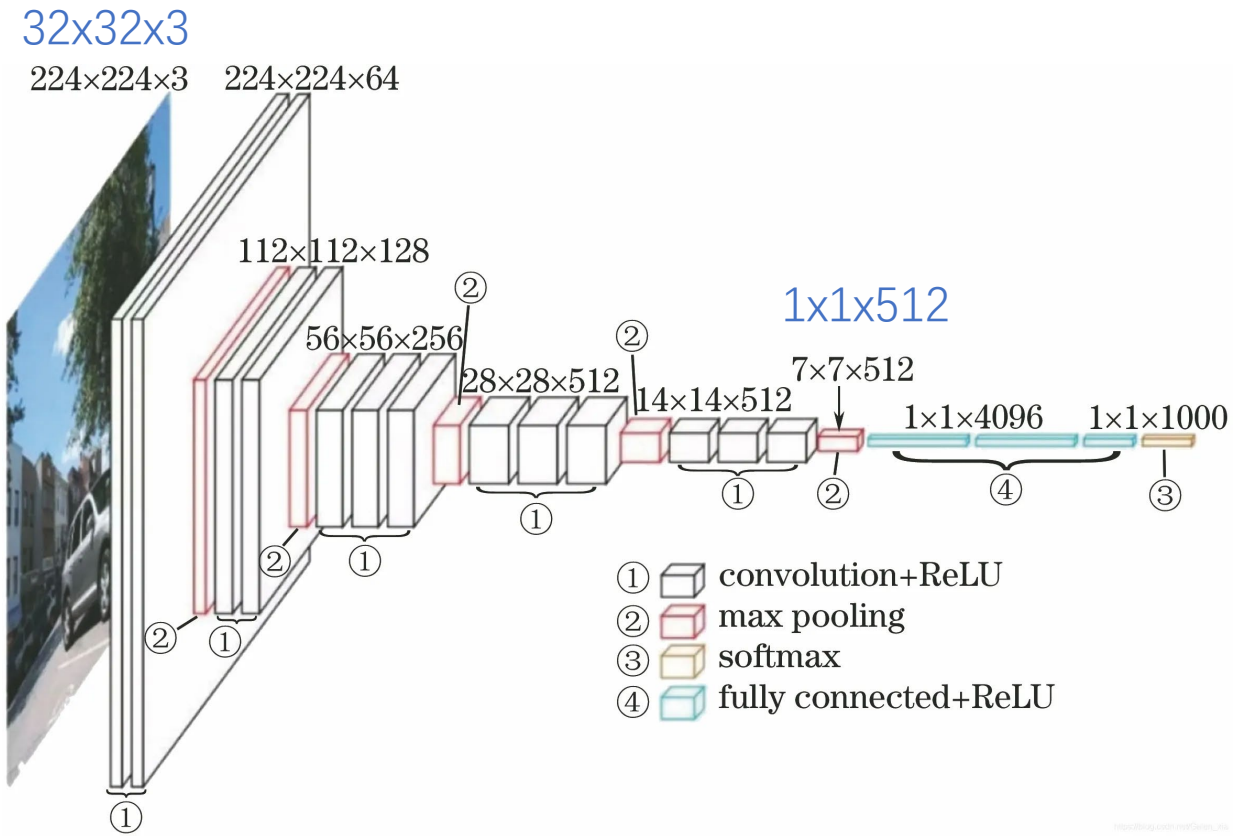
transforms.RandomCrop(224)

transforms.RandomResizedCrop(224)

transforms. Resize(224)

2. How to process different inputs?

(2)、如果确实需要修改网络，以CNN为例，通常需要修改全连接层。




```
self.classifier = nn.Sequential(
    nn.Linear(512*7*7, 4096),
    nn.ReLU(True),
    nn.Dropout(p=0.5),
    nn.Linear(4096, 4096),
    nn.ReLU(True),
    nn.Dropout(p=0.5),
    nn.Linear(4096, num_classes)
)
```

```
self.classifier = nn.Sequential(
    nn.Linear(512, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, num_class)
)
```

2. How to process different inputs?

(3)、使用自适应池化应对不同尺寸inputs。

nn.AdaptiveAvgPool2d((1, 1))

512x1x1 512x7x7 512x14x14 512x28x28 → 512x1x1

```
self.conv2_x = self._make_layer(block, 64, num_block[0], 1)
self.conv3_x = self._make_layer(block, 128, num_block[1], 2)
self.conv4_x = self._make_layer(block, 256, num_block[2], 2)
self.conv5_x = self._make_layer(block, 512, num_block[3], 2)
self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
self.fc = nn.Linear(512 * block.expansion, num_classes)
```