Task A3：卷积神经网络

2020212267 刘帅

---

0 - 目录说明

本次任务的最终文件夹结构应按如下方式组织：

```
A3
├── A3.py
├── runs#记录tensorboard的log信息
|    ├── Dec03_21-59-19_iZ2ze0s2a42sreq2efuopnZCNNwithDP #CNN+DP
|    ├── Dec03_23-50-55_iZ2ze0s2a42sreq2efuopnZCNN  #CNN baseline
|    ├── Dec04_02-10-38_iZ2ze0s2a42sreq2efuopnZCNNwithDPBN #CNN+DP+BN
|    ├── Dec04_02-29-49_iZ2ze0s2a42sreq2efuopnZCNNwithBN #CNN+BN
|    ├── Dec04_02-36-33_iZ2ze0s2a42sreq2efuopnZCNNwithDPBN10 #CNN+DP+BN+Kfold=10
|    ├── Dec04_02-37-47_iZ2ze0s2a42sreq2efuopnZCNNwithDP10 #CNN+DP+Kfold=10
|    ├── Dec04_02-37-58_iZ2ze0s2a42sreq2efuopnZCNNwithBN10 #CNN+BN+Kfold=10
|    ├── Dec04_02-38-54_iZ2ze0s2a42sreq2efuopnZCNNwithCNN10 #CNN+Kfold=10
|    ├── Dec03_19-38-18_iZ2ze0s2a42sreq2efuopnZ  #VGG10 baseline
|    ├── Dec03_23-29-00_iZ2ze0s2a42sreq2efuopnZvgg12 #VGG12
|    ├── Dec04_00-45-36_iZ2ze0s2a42sreq2efuopnZvgg15 #VGG15
|    ├── Dec04_00-49-03_iZ2ze0s2a42sreq2efuopnZvgg18 #VGG18
```

1 - 运行环境说明

本次实验内容在四卡服务器上进行。gcc version=9.4.0；Ubuntu=9.4.0；python=3.7.13；pytorch=1.9.0

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:0C.0 Off |                    0 |
| N/A   60C    P0   264W / 300W |   2942MiB / 16160MiB |     95%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:0D.0 Off |                    0 |
| N/A   60C    P0   253W / 300W |   2942MiB / 16160MiB |     95%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:0E.0 Off |                    0 |
| N/A   61C    P0   261W / 300W |   2942MiB / 16160MiB |     95%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:0F.0 Off |                    0 |
| N/A   55C    P0   242W / 300W |   2942MiB / 16160MiB |     95%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

2 - 代码说明

2.1 调用相关库 定义summarywriter

实例化tensorboard对象,并定义device，便于后续将数据及模型挪于gpu计算。

同时为了后面的cross-validate操作，从sklearn框架中调用KFold进行K折交叉验证。(通过资料查询，目前如果仅基于pytorch框架的话没有较好的实现方法。)

```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torch.optim as optim
import torchvision.models as model
from tqdm import tqdm
from torch.utils.tensorboard import SummaryWriter
from sklearn.model_selection import KFold

writer=SummaryWriter()
device=torch.device('cuda'if torch.cuda.is_available()else 'cpu')
```

2.2 数据集加载

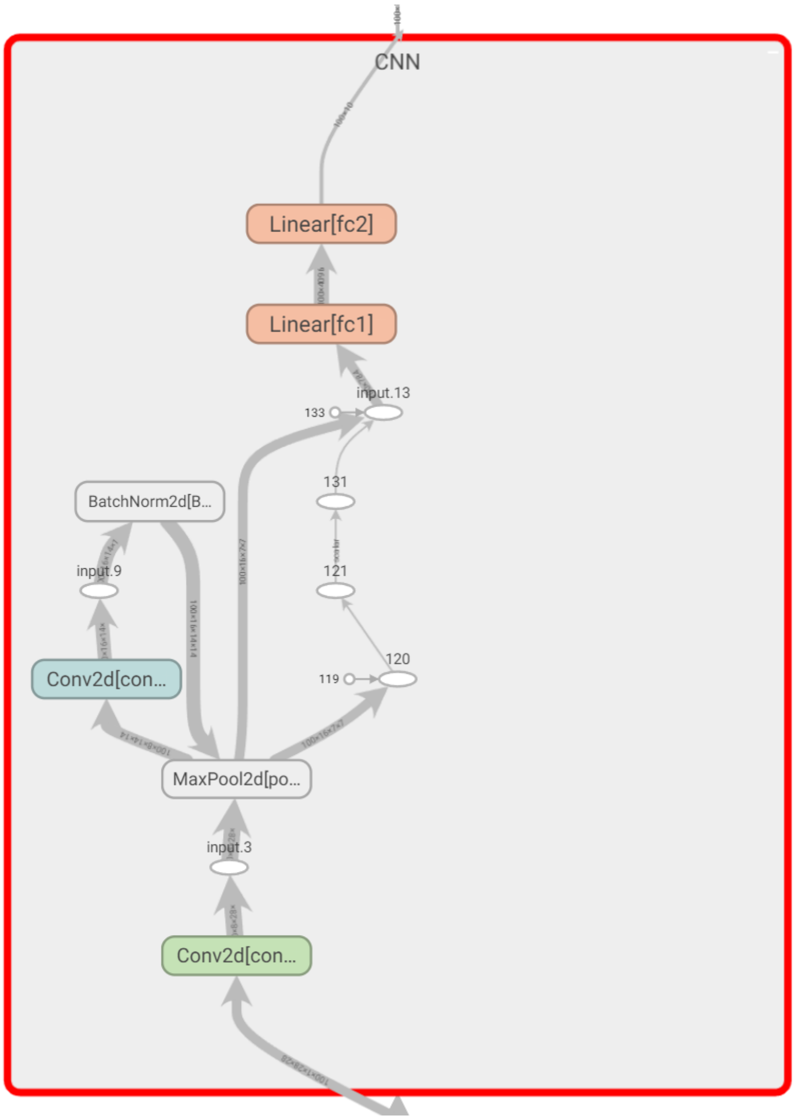
由于test_loader部分在后面仅做最终测试使用，故可以在此处定义。而针对trainloader和valloader的内容，为保证每个迭代轮次内数据的随机性，在后面经过一个kFold时再进行定义则更为稳妥。(不过本人针对该部分也进行了一些可视化实验，发现在kfold外部定义dataloader也是正确的)

```
device=torch.device('cuda'if torch.cuda.is_available()else 'cpu')

train_dataset=torchvision.datasets.MNIST('../data',train=True,download=False,

transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),torc
hvision.transforms.Normalize((0.11),(0.3))]))
test_dataset=torchvision.datasets.MNIST('../data',train=False,download=False,

transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),torc
hvision.transforms.Normalize((0.13,),(0.31))]))

test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=100,
num_workers=4)
```

2.3 网络结构定义

2.3.1 CNN结构

本实验采用CNN两层卷积层，两层池化层的CNN网络作为实验的baseline，网络的基本结构如下(该图利用tensorboard中add_graph函数进行输出，下文将再次提到，图中展示的为已加入dropout与bn层的CNN)




```
class CNN(nn.Module):
    def __init__(self,in_channels=1,num_classes=10):
        super(CNN,self).__init__()
        self.conv1=nn.Conv2d(in_channels=1,out_channels=8,kernel_size=
(3,3),stride=(1,1),padding=(1,1))#same convolution
        self.pool=nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))#14*14
        self.conv2=nn.Conv2d(in_channels=8,out_channels=16,kernel_size=
(3,3),stride=(1,1),padding=(1,1))
        self.BN2d=nn.BatchNorm2d(16)
        self.fc1=nn.Linear(16*7*7,4096)
        self.dp=nn.Dropout(0.5)
        self.fc2=nn.Linear(4096,num_classes)
    def forward(self,x,tag=None):
        x=F.relu(self.conv1(x))
        x=self.pool(x)
        x=F.relu(self.conv2(x))
        if(tag=='withDPBN'or tag=='withBN'):
            x=self.BN2d(x)
        x=self.pool(x)
        x=x.reshape(x.shape[0],-1)#64*784
        x=self.fc1(x)
        if(tag=='withDPBN'or tag=='withDP'):
            x=self.dp(x)
        x=self.fc2(x)
```

```
        return x
```

在结构中所定义卷积核为3*3，在每层输出加入激活函数relu，同时，针对卷积层和全连接层添加了batchnorm，并在全连接层加入dropout。

2.3.2 VGG 结构

本文采用的vgg结构重写了torchvision的源码。原本是想直接调用vgg16并更改其中的线性层并减少maxpool的数量进行实现的，在本人实验过程中，发现直接将层数设为None，会在调用mymodel的时候产生不可调用的错误，经分析，个人认为是由于虽然将torchvision给定的maxpool层设为了None，尽管在结构和数据形状上不存在问题，然而，module类不存在init()函数，因此遇到None时无法调用，个人当时的想法是将make_layer进行重写，识别到K层的None时直接跳过，将K+1层前移。然而，make_layer并不是VGG类下的方法。。（个人认为torchvision.model的源码在这里写得有些欠缺，可移植性较差）**或者也有可能有内置的方法？个人在翻阅相关资料后没有找到相关解法**

```
epoch=0:   0%|                                          | 0/3750 [00:00<?, ?it/s]
Traceback (most recent call last):
  File "A3.py", line 93, in <module>
    pred=mymodel(data)
  File "/home/liushuai/.conda/envs/liushuai/lib/python3.7/site-packages/torch/nn/modules/module.py"
, line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "A3.py", line 56, in forward
    output = self.model(x)
  File "/home/liushuai/.conda/envs/liushuai/lib/python3.7/site-packages/torch/nn/modules/module.py"
, line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/liushuai/.conda/envs/liushuai/lib/python3.7/site-packages/torchvision/models/vgg.py",
 line 49, in forward
    x = self.features(x)
  File "/home/liushuai/.conda/envs/liushuai/lib/python3.7/site-packages/torch/nn/modules/module.py"
, line 1051, in _call_impl
    return forward_call(*input, **kwargs)
  File "/home/liushuai/.conda/envs/liushuai/lib/python3.7/site-packages/torch/nn/modules/container.
py", line 139, in forward
    input = module(input)
TypeError: 'NoneType' object is not callable
```

```
class vggmodel(nn.Module):
    def __init__(self):
        super(vggmodel,self).__init__()
        VGG = model.vgg16(pretrained=True)
        VGG.features[0] = nn.Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        VGG.classifier[6] = nn.Linear(in_features=4096, out_features=10, bias=True)
        for i in range(len(VGG.features)):
            if isinstance(VGG.features[i], nn.MaxPool2d):
                VGG.features[i] = None
        self.model = VGG


    def forward(self, x):
        output = self.model(x)
        return output
```

于是，本人直接仿照VGG的源码内容在实验中进行VGG模型构建：

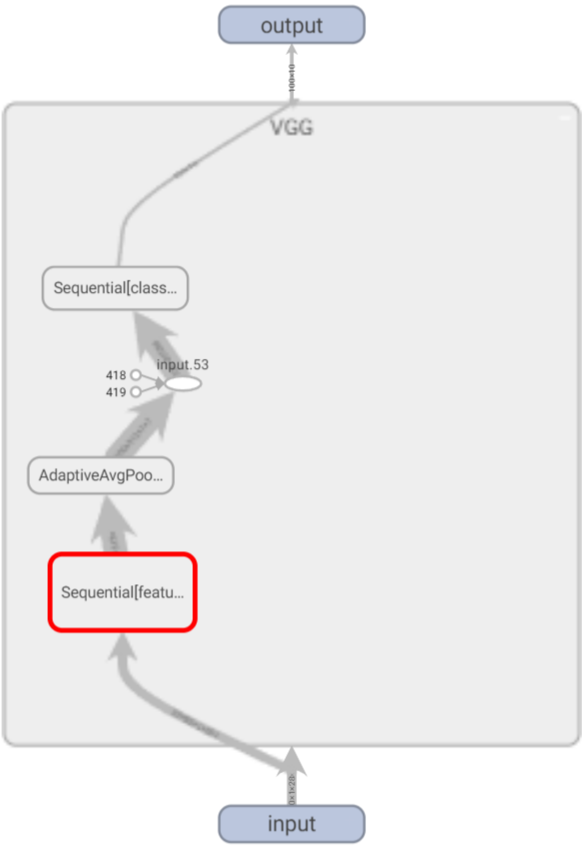
其中，共定义了四种vgg结构,分别为vgg10,vgg12,vgg15,vgg18。

```
cfgs = {
    'A': [64, 'M', 128,  256, 256, 'M', 512, 512,  512, 512],
    'B': [64, 64, 'M', 128, 128, 256, 256, 'M', 512, 512, 512, 512],
    'D': [64, 64, 'M', 128, 128, 256, 256, 256, 'M', 512, 512, 512, 512, 512,
512],
    'E': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 512, 512, 512, 512,
512, 512, 512, 512],
}
```

```
class VGG(nn.Module):

    def __init__(self, features, num_classes=10, init_weights=True):
        super(VGG, self).__init__()
        self.features = features
        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x,cfg=None):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 1
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
```

```
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
def _vgg(arch, cfg, batch_norm, pretrained, progress, **kwargs):
    if pretrained:
        kwargs['init_weights'] = False
    model = VGG(make_layers(cfgs[cfg], batch_norm=batch_norm),
**kwargs).to(device)
    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[arch],
                                              progress=progress)
        model.load_state_dict(state_dict)
    return model
def vgg_bn(pretrained=False, progress=True, **kwargs):
    return _vgg('vgg_bn', 'A', True, pretrained, progress, **kwargs)
```

同样，利用tensorboard直接描绘出VGG的结构示意图，由于体量较为庞大，在此仅对feature模块局部展示：



2.4 测试过程

模型测试部分的代码，在最终测试时使用

```
best_accuracy=0
def test_eval():
    global best_accuracy
    num_correct=0
    num_samples=0
    mymodel.eval()
    with torch.no_grad():
        for data,label in tqdm(test_loader):
            data=data.to(device=device)
            label=label.to(device=device)
            num_correct+=(mymodel(data).argmax(dim=1)==label).sum()
            num_samples+=mymodel(data).size(0)
    print('accuracy:',num_correct/num_samples)
    return num_correct/num_samples
```

2.5 配置及超参数定义

其中，config作为modelconfig的索引，便于后续的对比试验，n_splits表示交叉验证的折数，例如，此处的n_splits=4，代表每个迭代过程从训练集中选择1/4的内容作为测试集使用

```
config=0
modelconfig=['withDPBN','withDP','withBN',None]
in_channel=1
input_size=784
num_classes=10
learning_rate=0.001
batch_size=64
num_epochs=5
kfold=KFold(n_splits=4,shuffle=True)
```

2.6 模型定义

本次模型为CNN和vgg_bn,并调用DataParallel,在四张卡进行并行训练和推理。

```
mymodel=torch.nn.DataParallel(CNN(),device_ids=[0,1,2,3]).to(device)
# mymodel=torch.nn.DataParallel(vgg_bn(),device_ids=[0,1,2,3])
```

2.7 交叉验证训练流程

```
criterion=nn.CrossEntropyLoss()
optimizer=optim.Adam(mymodel.parameters(),lr=learning_rate)
step = 0
#定义loss和优化器内容，并定义全局变量step，k折交叉验证在每一轮都要进行epoch次训练，共一共-需要
k*epoch迭代轮次）
for fold,(train_ids,val_ids) in enumerate(kfold.split(train_dataset)):
    #随机对train和val数据集进行采样，保证数据的随机性
    train_subsampler=torch.utils.data.SubsetRandomSampler(train_ids)
    val_subsampler=torch.utils.data.SubsetRandomSampler(val_ids)
   # 定义dataloader
train_loader=torch.utils.data.DataLoader(train_dataset,batch_size=100,sampler=tr
ain_subsampler,num_workers=4)
    val_loader = torch.utils.data.DataLoader(train_dataset, batch_size=100,
sampler=val_subsampler, num_workers=4)


    #模拟一个轮次的数据输入，从而可以打印出model的模型，在tensorboard中可视化
    train_data_sample,_ = iter(train_loader).next()
    with writer:
        writer.add_graph(mymodel.module, train_data_sample.to(device))

    for epoch in range(num_epochs):
        loss=[]
        num_correct=0
        num_samples=0
        train_losses=0
        mymodel.train()
        for data,label in tqdm(train_loader,desc="epoch={}".format(epoch)):
            data=data.to(device=device)
            label=label.to(device=device)
            pred=mymodel(data,modelconfig[config])
            loss=criterion(pred,label)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```
            ---------------#每个epoch结束后，利用训练集对数据准度进行推测。----------
            with torch.no_grad():
                data=data.to(device=device)
                label=label.to(device=device)
                pred=mymodel(data,modelconfig[config])
                num_correct += (pred.argmax(dim=1) == label).sum()
                num_samples += pred.size(0)
                train_losses += loss.item()
        accuracy = num_correct / num_samples
        train_losses/=len(train_loader)
        print(f'fold:{fold},epoch:{epoch} Train - Loss:{train_losses} Accuracy:
{accuracy}')
        step+=1
        #利用writer在tensorboard可视化
        writer.add_scalars('trainloss',
{'kfold':fold,'epoch':epoch,'trainloss':train_losses},step)
        writer.add_scalars('trainacc', {'kfold': fold, 'epoch': epoch,
'trainacc': accuracy}, step)
        writer.add_text('trainlog',f'fold:{fold},epoch:{epoch} Train - Loss:
{train_losses} Accuracy:{accuracy}',step)
        writer.flush()
        writer.close()

            -------------#每个epoch结束后，利用测试集进行推测。------------
        num_correct=0
        num_samples=0
        val_losses=0
        with torch.no_grad():
            for data,label in tqdm(val_loader):
                data=data.to(device=device)
                label=label.to(device=device)
                pred=mymodel(data,modelconfig[config])
                loss=criterion(pred,label)
                num_correct+=(pred.argmax(dim=1)==label).sum()
                num_samples+=pred.size(0)
                val_losses+=loss.item()
        accuracy=num_correct/num_samples
        #利用验证集的结果对模型进行评估，并存储SOTA模型
        if num_correct / num_samples > best_accuracy:
            best_accuracy = num_correct / num_samples
        torch.save(mymodel, 'CNNmodel.pth')
        #利用writer在tensorboard可视化
        print(f'fold:{fold},epoch:{epoch} Val - Loss:{loss} Accuracy:
{accuracy}')
        writer.add_scalars('valloss',
{'kfold':fold,'epoch':epoch,'trainloss':train_losses},step)
        writer.add_scalars('valacc', {'kfold': fold, 'epoch': epoch, 'trainacc':
accuracy}, step)
        writer.add_text('testlog',f'fold:{fold},epoch:{epoch} Val - Loss:{loss}
Accuracy:{accuracy}',step)
        writer.flush()
        writer.close()
```

2.8 最终测试

```
acc=test_eval()
writer.add_scalar("loss",loss,epoch)
writer.add_scalar("acc",acc,epoch)
writer.add_text('test_accuracy', f'accuracy:{acc}')
writer.flush()
writer.close()
```

3. 数据可视化及分析

3.1 CNN训练结果

CNN的训练一共分为四个规格进行，本人在此处尝试加入BN后，按照原有batchsize=100的配置，test accuracy指标只有0.28，**远远低于不加入BN的结果**，而后将batchsize调到1000，发现正常，经分析，个人认为原理如下：

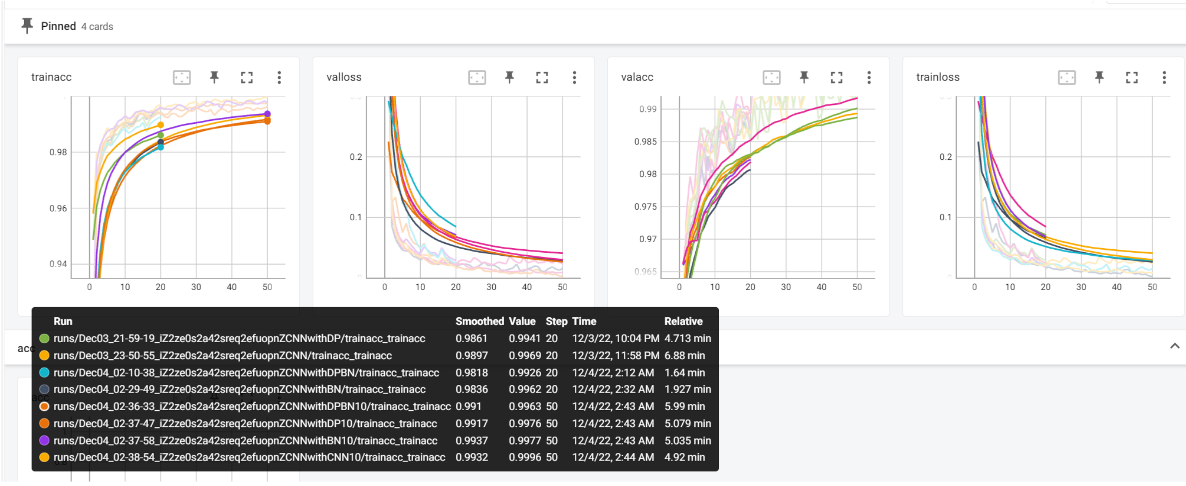
由于个人选用了dataparallel并行训练，实际每张卡分得的batchsize大小为25，因此batch过小，此时进行归一化效果很差，将batchsize调至1000后指标有所提升。然而，测试效果仍然不稳定，且表现并没有不加入BN的效果好。个人目前仍未解决的问题如下：

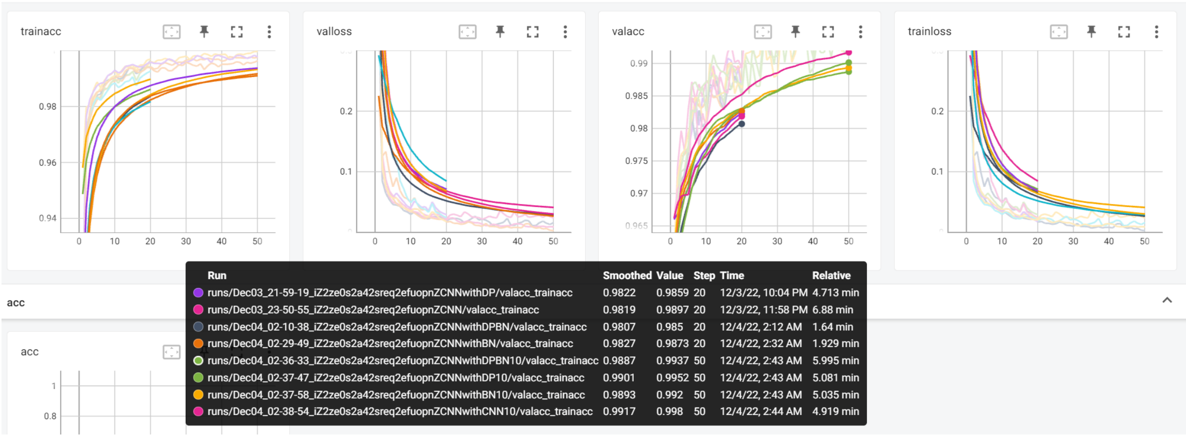
1、加入BN组实验下，在训练集和测试集的表现相差较大。经https://zhuanlan.zhihu.com/p/421878458搜索，已将batchsize大小调至相同，并且也切换到了model.eval（）模式，并且还加入了torch.no_grad进行双重约束，不知道问题究竟是什么原因

2、Batch Normalization，在train时不仅使用了当前batch的均值和方差，也使用了历史batch统计上的均值和方差，并做一个加权平均（momentum参数）。在test时，由于此时batchsize不一定一致，因此不再使用当前batch的均值和方差，仅使用历史训练时的统计值。在多卡训练问题中，可以采用SycnBN，其原理是因为每次迭代，输入被等分成多份，然后分别在不同的卡上前向（forward）和后向（backward）运算，并且求出梯度，在迭代完成后合并 梯度、更新参数，再进行下一次迭代。因为在前向和后向运算的时候，每个卡上的模型是单独运算的，所以相应的Batch Normalization 也是在卡内完成，所以实际BN所归一化的样本数量仅仅局限于卡内，相当于批量大小（batch-size）减小了。参考李沐老师的推文https://zhuanlan.zhihu.com/p/40496177。然而，该过程只适用于nvidia的DDP多卡并行，在正常的torch.dataparallel下不适用。

model	test accuracy
CNN+DP+BN(n_split=4)	accuracy:0.9355999827384949
CNN+DP(n_split=4)	accuracy:0.9828999638557434
CNN+BN(n_split=4)	accuracy:0.8575999736785889
CNN(n_split=4)	accuracy:0.9829999804496765
CNN+DP+BN(n_split=10)	accuracy:0.872999951839447
CNN+DP+BN(n_split=10)	accuracy:0.9853999614715576
CNN+BN(n_split=10)	accuracy:0.8789999485015869
CNN(n_split=10)	accuracy:0.9864999651908875

3.1.1 对比分析





通过图像对比可以得到以下结论

1、可以发现，10折交叉训练的效果略好于4折交叉训练，无论在训练指标还是测试指标上均有一个百分点的提升。

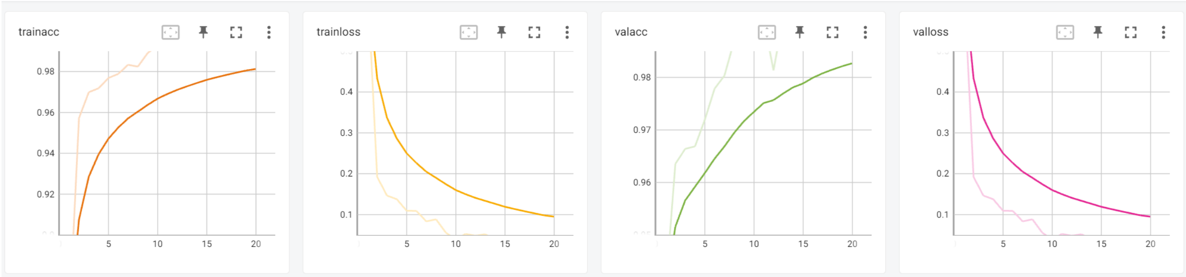
2、加入dropout和batchnorm的网络在训练前期的效果略好于CNNbaseline，而在后期有微弱劣势

3、加入dropout层的网络在训练起来收敛速度较慢，加入BatchNorm层的收敛速度较快

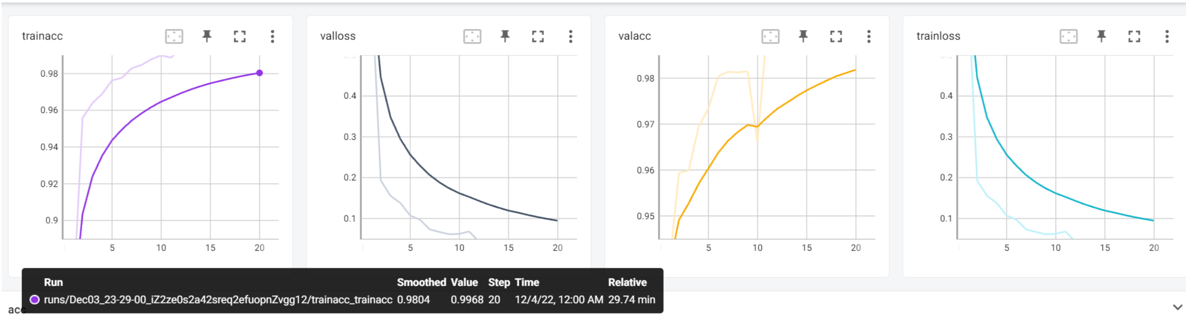
3.2 VGG训练结果

model	test accuracy
vgg10	accuracy:0.9921963855749638
vgg12	accuracy:0.993399977684021
vgg15	accuracy:0.9901999831199646
vgg18	accuracy:0.9934999942779541

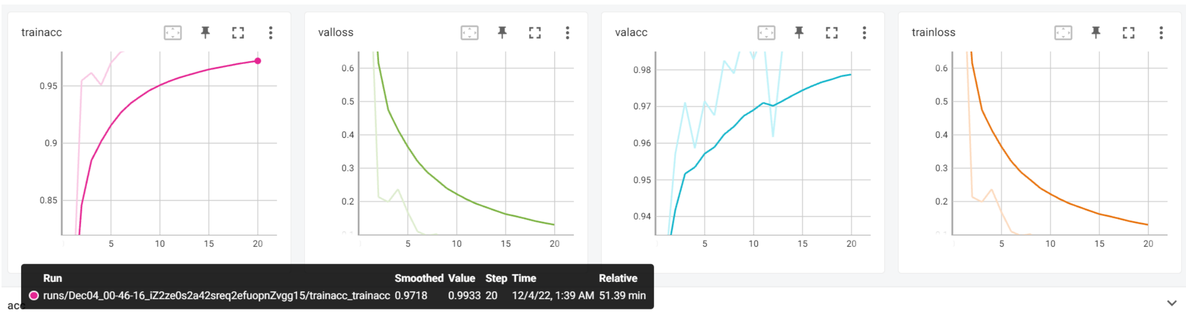
3.2.1 vgg10



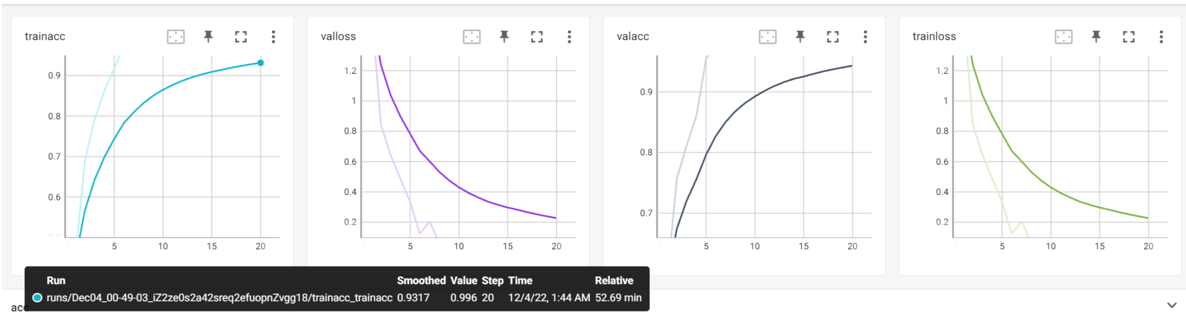
3.2.2 vgg12



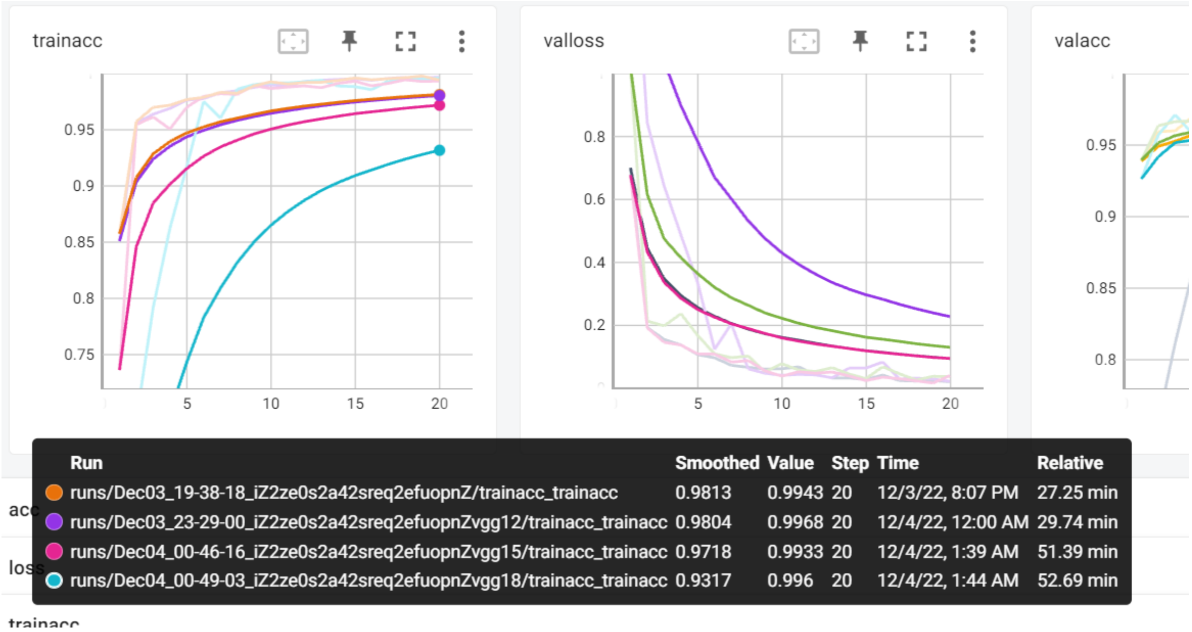
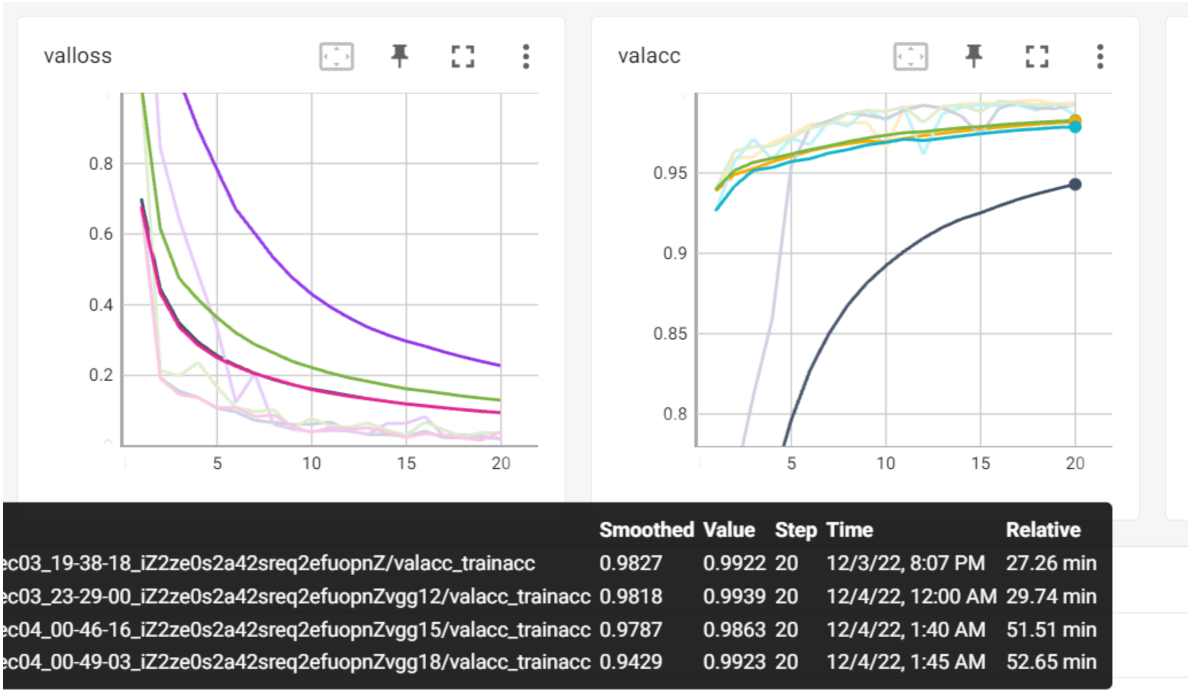
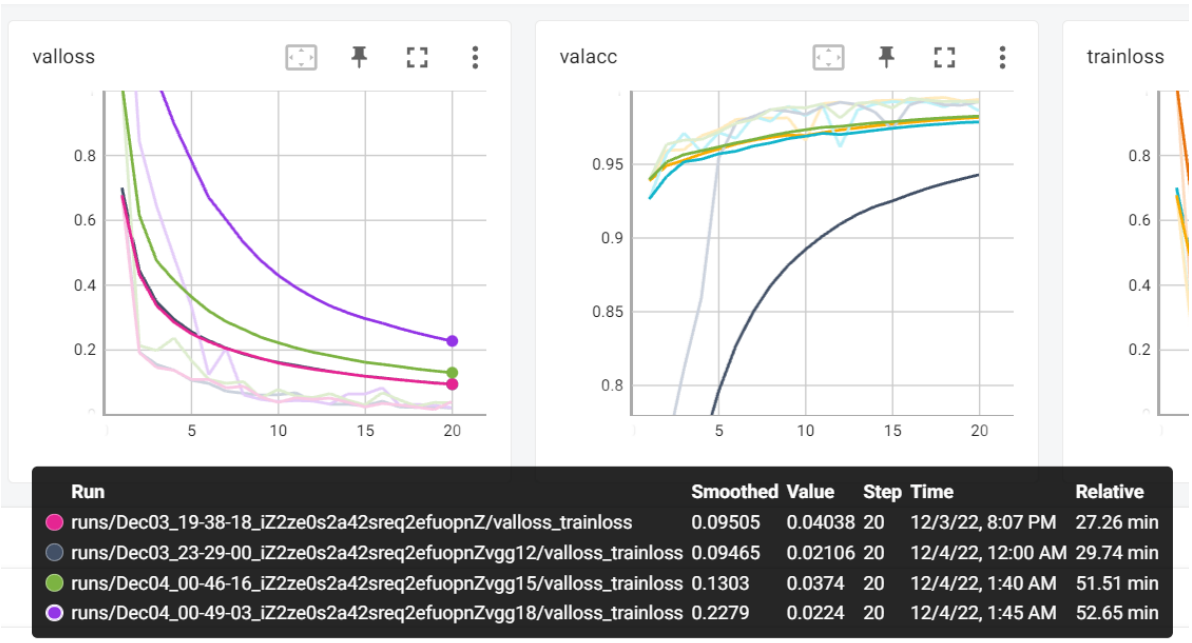
3.2.3 vgg15



3.2.4 vgg18

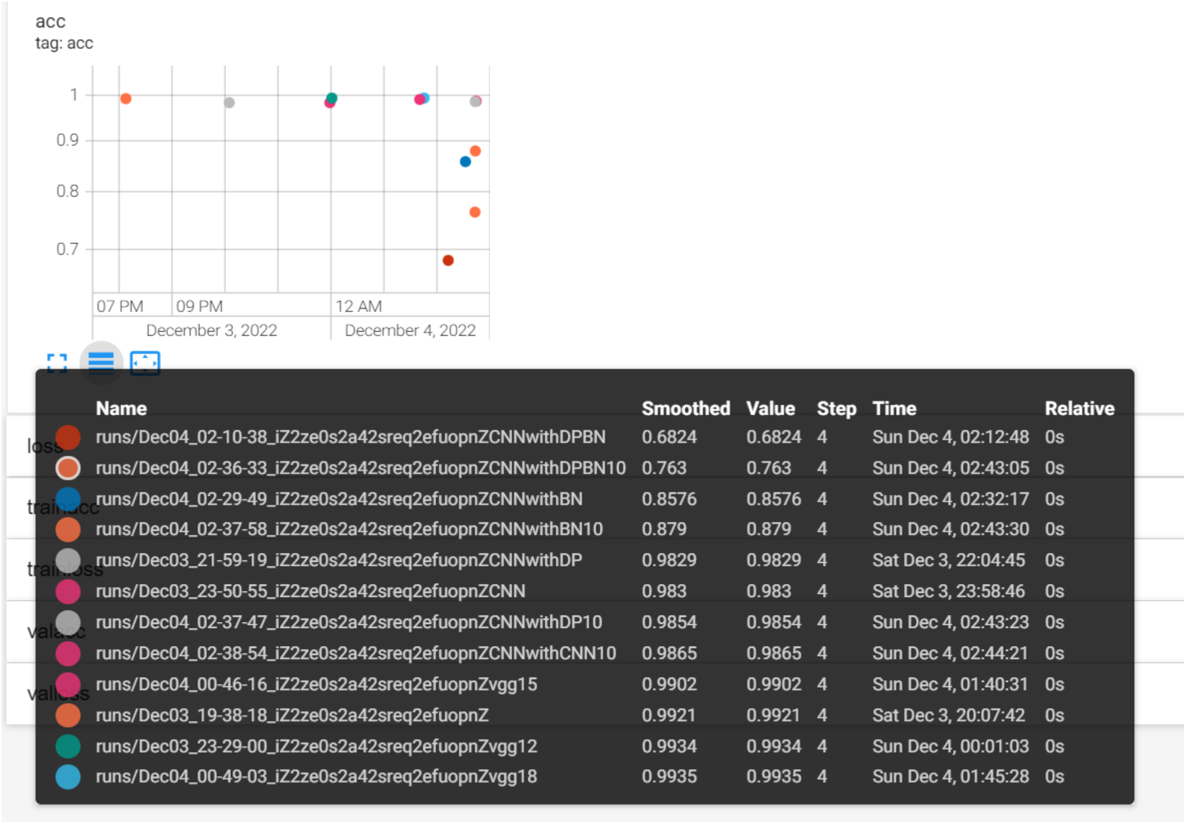


3.2.5 对比分析


trainacc
valloss
valacc
acc
loss
trainacc
Run Smoothed Value Step Time Relative
runs/Dec03_19-38-18_iZ2ze0s2a42sreq2efuopnZ/trainacc_trainacc 0.9813 0.9943 20 12/3/22, 8:07 PM 27.25 min
runs/Dec03_23-29-00_iZ2ze0s2a42sreq2efuopnZvgg12/trainacc_trainacc 0.9804 0.9968 20 12/4/22, 12:00 AM 29.74 min
runs/Dec04_00-46-16_iZ2ze0s2a42sreq2efuopnZvgg15/trainacc_trainacc 0.9718 0.9933 20 12/4/22, 1:39 AM 51.39 min
runs/Dec04_00-49-03_iZ2ze0s2a42sreq2efuopnZvgg18/trainacc_trainacc 0.9317 0.996 20 12/4/22, 1:44 AM 52.69 min

valloss
valacc
Smoothed Value Step Time Relative
ec03_19-38-18_iZ2ze0s2a42sreq2efuopnZ/valacc_trainacc 0.9827 0.9922 20 12/3/22, 8:07 PM 27.26 min
ec03_23-29-00_iZ2ze0s2a42sreq2efuopnZvgg12/valacc_trainacc 0.9818 0.9939 20 12/4/22, 12:00 AM 29.74 min
ec04_00-46-16_iZ2ze0s2a42sreq2efuopnZvgg15/valacc_trainacc 0.9787 0.9863 20 12/4/22, 1:40 AM 51.51 min
ec04_00-49-03_iZ2ze0s2a42sreq2efuopnZvgg18/valacc_trainacc 0.9429 0.9923 20 12/4/22, 1:45 AM 52.65 min

valloss
valacc
trainloss
Run Smoothed Value Step Time Relative
runs/Dec03_19-38-18_iZ2ze0s2a42sreq2efuopnZ/valloss_trainloss 0.09505 0.04038 20 12/3/22, 8:07 PM 27.26 min
runs/Dec03_23-29-00_iZ2ze0s2a42sreq2efuopnZvgg12/valloss_trainloss 0.09465 0.02106 20 12/4/22, 12:00 AM 29.74 min
runs/Dec04_00-46-16_iZ2ze0s2a42sreq2efuopnZvgg15/valloss_trainloss 0.1303 0.0374 20 12/4/22, 1:40 AM 51.51 min
runs/Dec04_00-49-03_iZ2ze0s2a42sreq2efuopnZvgg18/valloss_trainloss 0.2279 0.0224 20 12/4/22, 1:45 AM 52.65 min

观察vgg10，vgg12，vgg15，vgg18曲线变化情况，会发现，网络模型越大，其初始状态准确率越低，由于VGG网络在最初需要进行随机初始化，因此网络结构越复杂，其“错误参数越多”，学习的成本与代价越大，因此在前几个epoch的时候，在测试集上的准确率较低。但是，由于网络结构较复杂，故其可学习的参数更多，因此最终分类效果更好。同时我们可以观察到，vgg15的收敛速度相较而言最快，vgg10与vgg12收敛效果几乎相似，而vgg18最终loss仍停留在比较高的水平，个人认为此时vgg18仍未收敛，还有进一步提升空间。

3.3 全模型训练结果对比



Name	Smoothed	Value	Step	Time	Relative
runs/Dec04_02-10-38_iZ2ze0s2a42sreq2efuopnZCNNwithDPBN	0.6824	0.6824	4	Sun Dec 4, 02:12:48	0s
runs/Dec04_02-36-33_iZ2ze0s2a42sreq2efuopnZCNNwithDPBN10	0.763	0.763	4	Sun Dec 4, 02:43:05	0s
runs/Dec04_02-29-49_iZ2ze0s2a42sreq2efuopnZCNNwithBN	0.8576	0.8576	4	Sun Dec 4, 02:32:17	0s
runs/Dec04_02-37-58_iZ2ze0s2a42sreq2efuopnZCNNwithBN10	0.879	0.879	4	Sun Dec 4, 02:43:30	0s
runs/Dec03_21-59-19_iZ2ze0s2a42sreq2efuopnZCNNwithDP	0.9829	0.9829	4	Sat Dec 3, 22:04:45	0s
runs/Dec03_23-50-55_iZ2ze0s2a42sreq2efuopnZCNN	0.983	0.983	4	Sat Dec 3, 23:58:46	0s
runs/Dec04_02-37-47_iZ2ze0s2a42sreq2efuopnZCNNwithDP10	0.9854	0.9854	4	Sun Dec 4, 02:43:23	0s
runs/Dec04_02-38-54_iZ2ze0s2a42sreq2efuopnZCNNwithCNN10	0.9865	0.9865	4	Sun Dec 4, 02:44:21	0s
runs/Dec04_00-46-16_iZ2ze0s2a42sreq2efuopnZvgg15	0.9902	0.9902	4	Sun Dec 4, 01:40:31	0s
runs/Dec03_19-38-18_iZ2ze0s2a42sreq2efuopnZ	0.9921	0.9921	4	Sat Dec 3, 20:07:42	0s
runs/Dec03_23-29-00_iZ2ze0s2a42sreq2efuopnZvgg12	0.9934	0.9934	4	Sun Dec 4, 00:01:03	0s
runs/Dec04_00-49-03_iZ2ze0s2a42sreq2efuopnZvgg18	0.9935	0.9935	4	Sun Dec 4, 01:45:28	0s

4、更多trick

在训练过程中，本人尝试加入了混合精度训练amp，同时改变训练并行方式为DDP，从而进一步提高训练速度

```
from apex import amp
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
```