lab_demand

1. 使用 pytorch 或者 tensorflow 实现卷积神经网络 CNN，在 CIFAR-10 数据集上进行图片分类。研究 dropout、normalization、learning rate decay、卷积核大小、网络深度等超参数对分类性能的影响
2. 划分为训练集，验证集，测试集

- 验证集的作用：用不同的超参训练同一个训练集，用验证集上的效果比较，数据集较小，快速频繁反馈调超参
- 测试集只测试一次，作为调整好的模型的结果

3. 为便于直观比较，建议将验证集 loss 变化的曲线利用 **matplotlib** 绘图进行可视化。

如何构造复杂的RNN网络

1. VGG，（Conv+ReLU）*k+MaxPool，automation（更深的网络，smaller convolutional kernel
2. NiN，NiN block 输出通道数=标签类别的数量，全局平均 replace 全局最大，综合特征
3. GoogleNet，Inception block——先降低维数，用最简单的1*1提高这个过程的速度，再做卷积，全局平均
4. ResNet：拟合的是残差f(x)=h(x)-x，可用1*1降通道数

初始配置与结果

- 基于Lenet5，采用2层卷积+池化,后展开，用3层线性变换，(但提升了卷积通道数，减小了kernel_size，以获得更多更大的特征)效果不错

```
class Lenet5(nn.Module):
    def __init__(self):
        super(Lenet5,self).__init__()
        #nn.Sigmoid是一个类，先要赋值为对象
        self.acti=nn.ReLU()

 self.conv1=nn.Conv2d(in_channels=3,out_channels=16,kernel_size=5,stride=1)
 #in_channels为输入的通道
        #32-4=28
        self.pool1=nn.AvgPool2d(kernel_size=2,stride=2)
        #28/2=14

 self.conv2=nn.Conv2d(in_channels=16,out_channels=36,kernel_size=3,stride=1)
        #14-4=12
        self.pool2=nn.AvgPool2d(kernel_size=2,stride=2)
        #12/2=6
        self.fc1=nn.Linear(1296,128)
        self.fc2=nn.Linear(128,96)
        self.fc3=nn.Linear(96,10)

    def forward(self,x):
        #softmax用于多分类
        x=self.pool1(self.acti(self.conv1(x)))
        x=self.pool2(self.acti(self.conv2(x)))
        x=x.view(-1,1296)
        x=self.fc3(self.acti(self.fc2(self.acti(self.fc1(x)))))
```

```
        return x
```

```
[4, 8000]loss:1.000
[4,10000]loss:0.998
[4,12000]loss:0.986
Finished Training
[5, 2000]loss:0.883
[5, 4000]loss:0.872
[5, 6000]loss:0.891
[5, 8000]loss:0.873
[5,10000]loss:0.891
[5,12000]loss:0.888
Finished Training
[6, 2000]loss:0.778
[6, 4000]loss:0.784
[6, 6000]loss:0.783
[6, 8000]loss:0.797
[8, 4000]loss:0.613
[8, 6000]loss:0.625
[8, 8000]loss:0.663
[8,10000]loss:0.664
[8,12000]loss:0.667
Finished Training
[9, 2000]loss:0.520
[9, 4000]loss:0.554
[9, 6000]loss:0.574
[9, 8000]loss:0.589
[9,10000]loss:0.587
[9,12000]loss:0.595
Finished Training
[10, 2000]loss:0.456
[10, 4000]loss:0.496
[10, 6000]loss:0.501
[10, 8000]loss:0.520
[10,10000]loss:0.543
[10,12000]loss:0.547
Finished Training
Accuracy of the network on the 10000 test images: 68 %
```

- PS:应该用验证集评估，开始没将训练集划分为训练集和验证集

如何优化(简单网络)

> 1. 验证集和测试集
> 2. 正则化项
> 3. BN(批归一化)，IN(层归一化)，GN(同一层，批道组，组批道数|通道数)
> 4. Dropout：训练时，在每一个批中，对每一层，神经元以概率p=0
> 5. Dropconnect
> 6. **(Early stop)**提前终止，验证集准确率下降(训练集损失仍然下降)，而非测试集
> 7. 调超参
> 8. append noise
> 9. 标签平滑

1. 验证集和测试集

- 将trainset划分为trainset+validset
- torch.utils.data.random_split(dataset1,[train_size,valid_size])的参数需为int

```
dataset1=torchvision.datasets.CIFAR10(root='./data',train=True,download=False,tr
ansform=transform)
train_size=int(0.8*(len(dataset1)))
valid_size=len(dataset1)-train_size
trainset,validset=torch.utils.data.random_split(dataset1,
[train_size,valid_size])
trainloader=torch.utils.data.DataLoader(trainset,batch_size=4,shuffle=True,num_w
orkers=0)
validloader=torch.utils.data.DataLoader(validset,batch_size=4,shuffle=True,num_w
orkers=0)
```

2. 正则化项

- If we want running_loss only recording difference between outputs and lablels,

  we shoud running_loss+=criterion(outputs,labels).item() rather than loss.item()

```
        l2_lambda = 0.01
        l2_regularization = sum(torch.sum(torch.pow(param, 2)) for param in
net.parameters())
...
        running_loss+=criterion(outputs,labels).item()
        loss=criterion(outputs,labels)+l2_lambda*l2_regularization
        loss.backward()
        optimizer.step()
```

3. Dropout

- 在输入层和隐藏层都使用Dropout
- 可以调高学习速率和冲量，leaning_rate*=10，momentum=0.9~0.99

```
        self.dropout=nn.Dropout(0.5)
...
        x=self.dropout(self.bn1(self.conv1(x)))    #激活前批正则化
...
        x=self.dropout(self.bn2(self.conv2(x)))
```

4. 批归一化

- 激活前批正则化
- BatchNormxd的num_features=通道数，≠批样本数
- 可以选择较大的初始学习率
- no ~~for epoch in range(epoch)~~ !

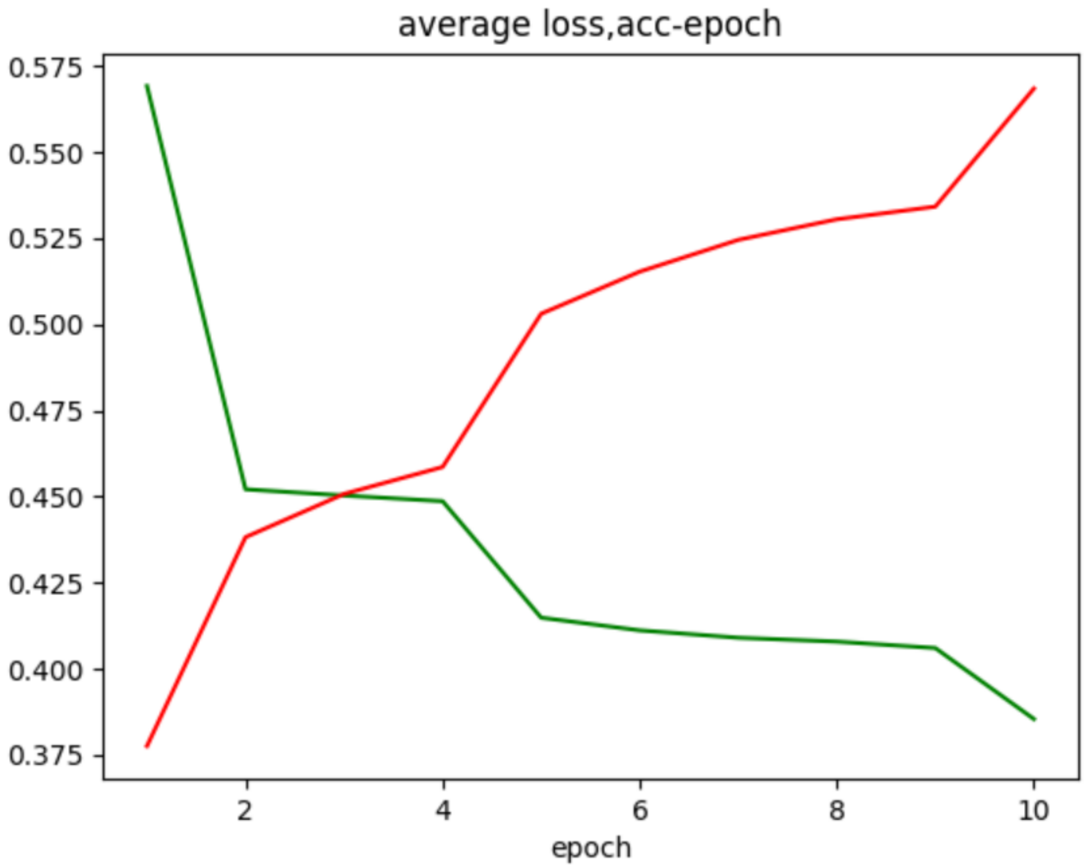
```
        self.bn1=nn.BatchNorm2d(num_features=6)
        self.bn2=nn.BatchNorm2d(num_features=16)
        self.bn3=nn.BatchNorm1d(num_features=120)
        self.bn4=nn.BatchNorm1d(num_features=84)

    def forward(self,x):
        #softmax用于多分类
        x=self.bn1(self.conv1(x))     #激活前批正则化
        x=self.pool1(self.acti(x))  #4*6*28*28
        x=self.bn2(self.conv2(x))
        x=self.pool2(self.acti(x))
        x=x.view(-1,16*5*5)

x=self.fc3(self.bn4(self.acti(self.fc2(self.acti(self.bn3(self.fc1(x)))))))
```

- valid/test 输出要另外写

用来Dropout之后验证精度反而下降了

```
epoch:1;     loss:0.569
epoch:2;     loss:0.452
epoch:3;     loss:0.450
epoch:4;     loss:0.449
epoch:5;     loss:0.415
epoch:6;     loss:0.411
epoch:7;     loss:0.409
epoch:8;     loss:0.408
epoch:9;     loss:0.406
epoch:10;     loss:0.385
Accuracy of the network on the 10000 test images: 50 %
```

但loss和acc的趋势是对的



5. Drop Connect

- DropConnect的朴素实现：

  对一个权重矩阵，其每个元素以p概率更改=0，Hadamard积:mat1*mat2

```
import torch
p = 0.5;    h=5;    w=6
dc = (torch.rand(h, w) < p).float()
#将权重矩阵*dc，make dropconnect
```

6. gradient clip

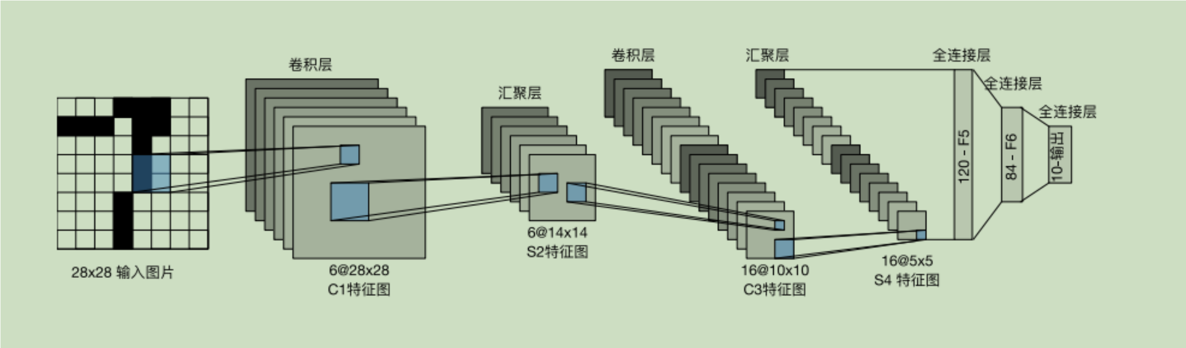
```
        nn.utils.clip_grad_norm_(net.parameters(), 1)   #gradient clip
```

7. 调超参

开始按照Lenet5，各层参数如下：

```
class Lenet5(nn.Module):
    def __init__(self):
        super(Lenet5,self).__init__()
        #nn.Sigmoid是一个类，先要赋值为对象
        self.acti=nn.Sigmoid()

self.conv1=nn.Conv2d(in_channels=3,out_channels=16,kernel_size=5,stride=1)
#in_channels为输入的通道
        #32-4=28
        self.pool1=nn.AvgPool2d(kernel_size=2,stride=2)
        #28/2=14
```

```
self.conv2=nn.Conv2d(in_channels=6,out_channels=16,kernel_size=5,stride=1)
        #14-4=10
        self.pool2=nn.AvgPool2d(kernel_size=2,stride=2)
        #10/2=5
        self.fc1=nn.Linear(16*5*5,120)
        self.fc2=nn.Linear(120,84)
        self.fc3=nn.Linear(84,10)
```



- 不做优化，lenet复刻参数，训练效果

```
[3,12000]loss:2.306
Finished Training
[4, 2000]loss:2.305
[4, 4000]loss:2.305
[4, 6000]loss:2.305
[4, 8000]loss:2.305
[4,10000]loss:2.305
[4,12000]loss:2.305
Finished Training
[5, 2000]loss:2.305
[5, 4000]loss:2.304
[5, 6000]loss:2.304
[5, 8000]loss:2.304
[5,10000]loss:2.304
[5,12000]loss:2.304
Finished Training
[6, 2000]loss:2.304
[6, 4000]loss:2.304
[6, 6000]loss:2.304
[6, 8000]loss:2.304
[6,10000]loss:2.304
[6,12000]loss:2.304
Finished Training
[7, 2000]loss:2.304
[7, 4000]loss:2.304
[7, 6000]loss:2.304
[7, 8000]loss:2.304
[7,10000]loss:2.304
[7,12000]loss:2.303
Finished Training
[8, 2000]loss:2.303
[8, 4000]loss:2.303
[8, 6000]loss:2.302
[8, 8000]loss:2.303
[8,10000]loss:2.302
[8,12000]loss:2.301
Finished Training
[9, 2000]loss:2.299
[9, 4000]loss:2.296
[9, 6000]loss:2.289
[9, 8000]loss:2.262
[9,10000]loss:2.196
[9,12000]loss:2.122
Finished Training
[10, 2000]loss:2.092
[10, 4000]loss:2.084
[10, 6000]loss:2.076
[10, 8000]loss:2.056
[10,10000]loss:2.062
[10,12000]loss:2.055
Finished Training
```

- 而后在卷积层，增加输出通道数，减小kernel_size，以生成更多更大的图像特征，效果依旧，一连串的2.30x

```
[8, 4000]loss:2.303
[8, 6000]loss:2.303
[8, 8000]loss:2.303
[8,10000]loss:2.303
[8,12000]loss:2.303
Finished Training
[9, 2000]loss:2.302
[9, 4000]loss:2.302
[9, 6000]loss:2.302
[9, 8000]loss:2.301
[9,10000]loss:2.300
[9,12000]loss:2.298
Finished Training
[10, 2000]loss:2.290
[10, 4000]loss:2.256
[10, 6000]loss:2.174
[10, 8000]loss:2.112
[10,10000]loss:2.085
[10,12000]loss:2.084
Finished Training
Accuracy of the network on the 10000 test images: 20 %
```

- 采用Relu作为激活函数后，效果大增

```
[7, 2000]loss:0.663
[7, 4000]loss:0.667
[7, 6000]loss:0.708
[7, 8000]loss:0.691
[7,10000]loss:0.698
[7,12000]loss:0.714
Finished Training
[8, 2000]loss:0.579
[8, 4000]loss:0.597
[8, 6000]loss:0.604
[8, 8000]loss:0.623
[8,10000]loss:0.657
[8,12000]loss:0.632
Finished Training
[9, 2000]loss:0.516
[9, 4000]loss:0.535
[9, 6000]loss:0.557
[9, 8000]loss:0.554
[9,10000]loss:0.575
[9,12000]loss:0.588
Finished Training
[10, 2000]loss:0.433
[10, 4000]loss:0.474
[10, 6000]loss:0.495
[10, 8000]loss:0.520
[10,10000]loss:0.525
[10,12000]loss:0.512
Finished Training
Accuracy of the network on the 10000 test images: 69 %
```

- 采用ReLU激活，即使用Lenet的较少的特征(通道)数，效果仍较好，可认为RELU起主要作用

```
Finished Training
[3, 2000]loss:1.318
[3, 4000]loss:1.302
[3, 6000]loss:1.278
[3, 8000]loss:1.285
[3,10000]loss:1.268
[3,12000]loss:1.231
Finished Training
[4, 2000]loss:1.172
[4, 4000]loss:1.169
[4, 6000]loss:1.199
[4, 8000]loss:1.190
[4,10000]loss:1.176
[4,12000]loss:1.174
Finished Training
[5, 2000]loss:1.088
[7, 2000]loss:0.985
[7, 4000]loss:0.987
[7, 6000]loss:0.975
[7, 8000]loss:0.997
[7,10000]loss:1.005
[7,12000]loss:0.982
Finished Training
[8, 2000]loss:0.913
[8, 4000]loss:0.926
[8, 6000]loss:0.932
[8, 8000]loss:0.937
[8,10000]loss:0.988
[8,12000]loss:0.965
Finished Training
[9, 2000]loss:0.880
[9, 4000]loss:0.895
[9, 6000]loss:0.900
[9, 8000]loss:0.917
[9,10000]loss:0.938
[9,12000]loss:0.906
Finished Training
[10, 2000]loss:0.827
[10, 4000]loss:0.857
[10, 6000]loss:0.881
[10, 8000]loss:0.883
[10,10000]loss:0.872
[10,12000]loss:0.891
Finished Training
Accuracy of the network on the 10000 test images: 62 %
```

- 因此后续卷积层采用较多的通道，提升表达能力，采用ReLU作为激活函数

做了优化1-5+optim=sigmoid

- Lenet复刻参数+sigmoid为优化函数

  output loss(with 正则化项)

```
Finished Training
[25,  2000]loss:2.325
[25,  4000]loss:2.321
[25,  6000]loss:2.319
[25,  8000]loss:2.318
[25,10000]loss:2.319
Finished Training
[26,  2000]loss:2.320
[26,  4000]loss:2.321
[26,  6000]loss:2.320
[26,  8000]loss:2.318
[26,10000]loss:2.322
Finished Training
[27,  2000]loss:2.324
[27,  4000]loss:2.322
[27,  6000]loss:2.320
[27,  8000]loss:2.320
[27,10000]loss:2.324
Finished Training
[28,  2000]loss:2.320
[28,  4000]loss:2.317
[28,  6000]loss:2.321
[28,  8000]loss:2.318
[28,10000]loss:2.320
Finished Training
[29,  2000]loss:2.317
[29,  4000]loss:2.324
[29,  6000]loss:2.323
[29,  8000]loss:2.319
[29,10000]loss:2.326
Finished Training
[30,  2000]loss:2.311
[30,  4000]loss:2.311
[30,  6000]loss:2.312
[30,  8000]loss:2.312
[30,10000]loss:2.313
Finished Training
```

- 复刻参数+relu为激活函数，类似，表明单纯用relu替代sigmoid，不提升特征数无效

8.lr

在PyTorch中，可以使用学习率衰减方法来调整训练时的学习率。PyTorch提供了多种学习率衰减方法，可以通过修改优化器的参数来实现。

以下是使用PyTorch实现学习率衰减的示例：

```
import torch.optim as optim
# 定义优化器
optimizer = optim.SGD(model.parameters(), lr=0.1)

# 定义学习率衰减方法：StepLR
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

# 在训练时更新学习率
for epoch in range(num_epochs):
    # ...

    # 更新optimizer中的学习率
    scheduler.step()

    # ...
```

在以上示例中，首先定义了一个`SGD`优化器，并设置学习率为`0.1`。接着，定义了一个学习率衰减方法`StepLR`，其中`step_size`参数表示学习率衰减的步长，`gamma`参数表示学习率衰减的比例。最后，在训练时，在每个epoch之后调用`optim.lr_scheduler.StepLR()`方法来更新优化器的学习率。

除了`StepLR`方法外，PyTorch还提供了其他的学习率衰减方法，如`ReduceLROnPlateau`(当监测指标未改善时降低学习率)、`CosineAnnealingLR`(余弦退火)、`MultiStepLR`(多步学习率衰减)等，可以根据具体需求选择合适的方法。

9. momentum

$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha g_t = -\alpha\sum_{\tau=1}^{t}\rho^{t-\tau}g_\tau$

$\beta_1 = 0.99$就相当于是之前所有梯度的累加，

虽然梯度裁剪可以避免梯度爆炸或消失，但可能急剧减慢训练速度

1. 有了weight decay，可以不要梯度裁剪，不然训练地太慢了

   不过出现了nan，可以调小初始学习率(或进行裁剪)

以2层全连接层的网络为例，

1. lr=0.001,momentum=0.99,

不进行梯度裁剪

```
root@autodl-container-793811833c-c2928d02:~# python untitled.py
Lenet5(
  (acti): ReLU()
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 36, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1296, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=96, bias=True)
  (fc3): Linear(in_features=96, out_features=10, bias=True)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm1d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
epoch:1;    loss:nan
```

进行梯度裁剪

```
root@autodl-container-793811833c-c2928d02:~# python untitled.py
Lenet5(
  (acti): ReLU()
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 36, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1296, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=96, bias=True)
  (fc3): Linear(in_features=96, out_features=10, bias=True)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm1d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
epoch:1;    loss:13.259
```

- 可见梯度裁剪能有效避免梯度爆炸或消失

lr=0.01,momentum=0.9，no clip

```
root@autodl-container-793811833c-c2928d02:~# python untitled.py
Lenet5(
  (acti): ReLU()
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 36, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1296, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=96, bias=True)
  (fc3): Linear(in_features=96, out_features=10, bias=True)
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm1d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)

epoch:1;    loss:0.563
epoch:2;    loss:0.453
epoch:3;    loss:0.450
epoch:4;    loss:0.448
```

- 可见momentum是一个非常重要的参数，调太大会导致nan,
- (卷积+池化层)*2+2层全连接层+dropout如下配置的结果

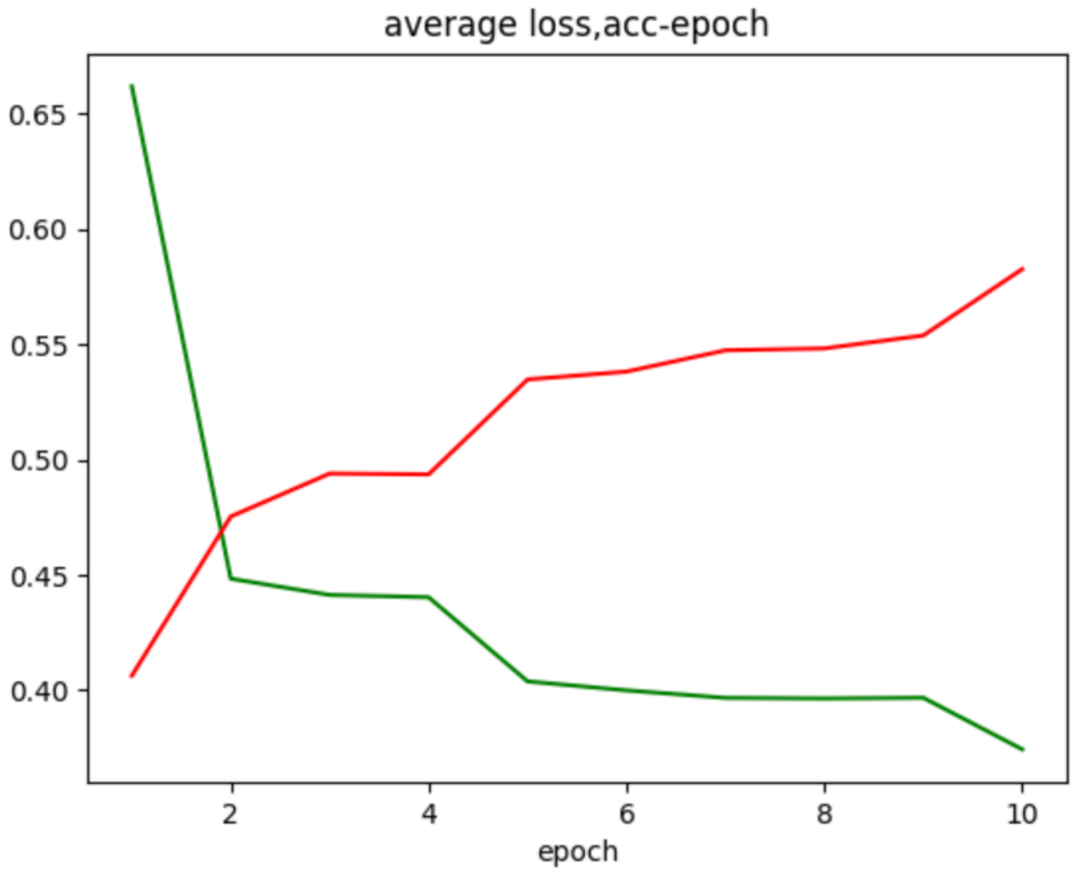
```
    def forward(self,x,p):
        #softmax用于多分类
        dropout=nn.Dropout(p)
        x=dropout(self.conv1(x))     #激活前批正则化
        x=self.pool1(self.acti(x))   #4*6*28*28
        x=dropout(self.conv2(x))
        x=self.pool2(self.acti(x))
        x=x.view(-1,1296)
        x=self.fc4(self.acti(self.fc1(x)))
        return x
    nn.utils.clip_grad_norm_(net.parameters(), 1)   #gradient clip

    optimizer=optim.SGD(net.parameters(),lr=0.01,momentum=0.99)
...
    nn.utils.clip_grad_norm_(net.parameters(), 1)   #gradient clip
```

```
epoch:1;     loss:0.592
epoch:2;     loss:0.433
epoch:3;     loss:0.426
epoch:4;     loss:0.425
epoch:5;     loss:0.387
epoch:6;     loss:0.382
epoch:7;     loss:0.377
epoch:8;     loss:0.374
epoch:9;     loss:0.373
epoch:10;     loss:0.351
Accuracy of the network on the 10000 test images: 56 %
```

- 卷积层后添上 BatchNorm2d后的结果

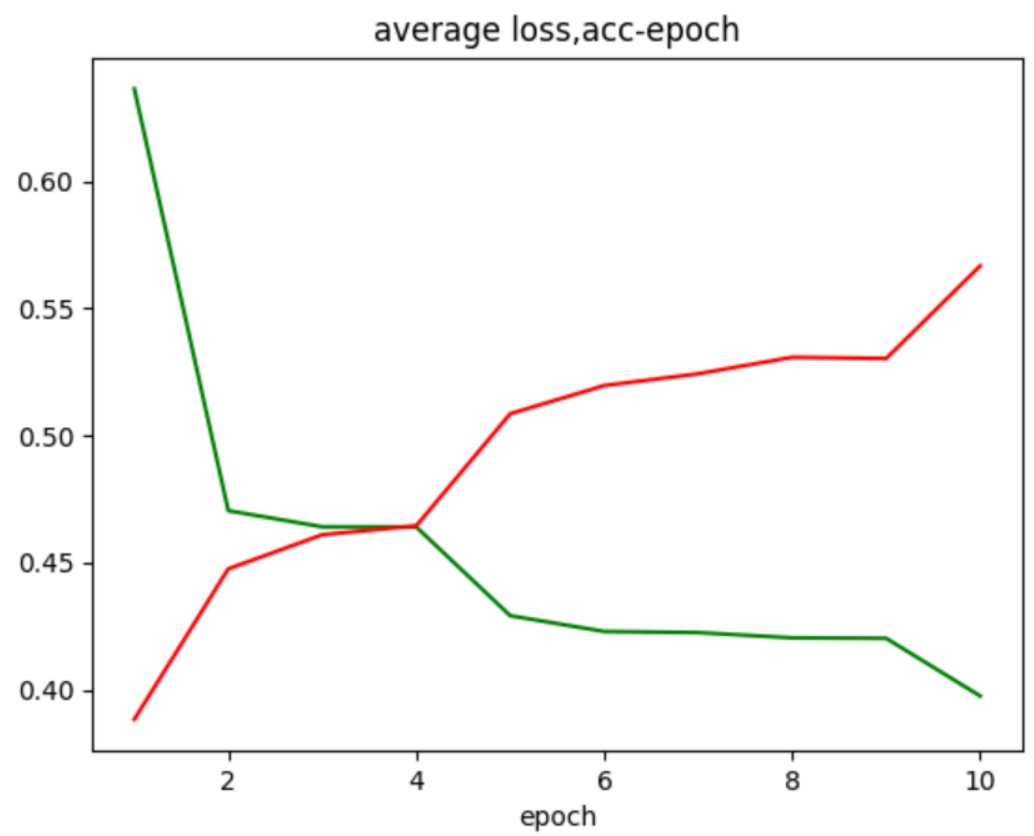
```
    def forward(self,x,p):
        #softmax用于多分类
        dropout=nn.Dropout(p)
        x=dropout(self.bn1(self.conv1(x)))     #激活前批正则化
        x=self.pool1(self.acti(x))   #4*6*28*28
        x=dropout(self.bn2(self.conv2(x)))
        x=self.pool2(self.acti(x))
        x=x.view(-1,1296)
        x=self.fc4(self.acti(self.fc1(x)))
        return x

    def dropforward(self,x,p):
        x=self.bn1(self.conv1(x))*(1-p)     #激活前批正则化
        x=self.pool1(self.acti(x))   #4*6*28*28
        x=self.bn2(self.conv2(x))*(1-p)
        x=self.pool2(self.acti(x))
        x=x.view(-1,1296)
        x=self.fc4((self.acti(self.fc1(x))))
        return x
```

```
epoch:1;     loss:0.662
epoch:2;     loss:0.448
epoch:3;     loss:0.441
epoch:4;     loss:0.440
epoch:5;     loss:0.404
epoch:6;     loss:0.400
epoch:7;     loss:0.397
epoch:8;     loss:0.396
epoch:9;     loss:0.397
epoch:10;     loss:0.374
Accuracy of the network on the 10000 valid images: 60 %
```



- 卷积+池化层后，2层全连接层与3层全连接层(结果和代码如下)的对比

```
epoch:1;     loss:0.636
epoch:2;     loss:0.470
epoch:3;     loss:0.464
epoch:4;     loss:0.464
epoch:5;     loss:0.429
epoch:6;     loss:0.423
epoch:7;     loss:0.423
epoch:8;     loss:0.420
epoch:9;     loss:0.420
epoch:10;     loss:0.398
Accuracy of the network on the 10000 test images: 58 %
```



- 三层全连接层竟然比2层效果差，可能是训练周期不够

```python
    def forward(self,x,p):
        #softmax用于多分类
        dropout=nn.Dropout(p)
        x=dropout(self.bn1(self.conv1(x)))      #激活前批正则化
        x=self.pool1(self.acti(x))   #4*6*28*28
        x=dropout(self.bn2(self.conv2(x)))
        x=self.pool2(self.acti(x))
        x=x.view(-1,1296)
        x=self.fc3(self.acti(self.fc2(self.acti(self.fc1(x)))))
        return x

    def dropforward(self,x,p):
        x=self.bn1(self.conv1(x))*(1-p)      #激活前批正则化
        x=self.pool1(self.acti(x))   #4*6*28*28
        x=self.bn2(self.conv2(x))*(1-p)
```

```
        x=self.pool2(self.acti(x))
        x=x.view(-1,1296)
        x=self.fc3(self.acti(self.fc2(self.acti(self.fc1(x)))))
        return x
```

简单网络

样板1

batchnorm+dropout(p=0.5)+3层全连接层+L2正则化+梯度裁剪，valid accuracy=64%

- running_loss=criterion(outputs,labels)+l2_lambda*l2_regularization

```
)
epoch:1;     loss:0.633
epoch:2;     loss:0.474
epoch:3;     loss:0.468
epoch:4;     loss:0.465
epoch:5;     loss:0.431
epoch:6;     loss:0.424
epoch:7;     loss:0.422
epoch:8;     loss:0.420
epoch:9;     loss:0.419
epoch:10;     loss:0.399
epoch:11;     loss:0.396
epoch:12;     loss:0.392
epoch:13;     loss:0.393
epoch:14;     loss:0.391
epoch:15;     loss:0.378
epoch:16;     loss:0.376
epoch:17;     loss:0.375
epoch:18;     loss:0.373
epoch:19;     loss:0.372
epoch:20;     loss:0.365
epoch:21;     loss:0.362
epoch:22;     loss:0.362
epoch:23;     loss:0.362
epoch:24;     loss:0.362
epoch:25;     loss:0.356
epoch:26;     loss:0.357
epoch:27;     loss:0.357
epoch:28;     loss:0.357
epoch:29;     loss:0.355
epoch:30;     loss:0.352
epoch:31;     loss:0.352
epoch:32;     loss:0.352
epoch:33;     loss:0.352
epoch:34;     loss:0.350
epoch:35;     loss:0.349
epoch:36;     loss:0.350
epoch:37;     loss:0.349
epoch:38;     loss:0.350
epoch:39;     loss:0.348
epoch:40;     loss:0.349
epoch:41;     loss:0.349
epoch:42;     loss:0.349
epoch:43;     loss:0.348
epoch:44;     loss:0.349
```

```
epoch:45;    loss:0.347
epoch:46;    loss:0.348
epoch:47;    loss:0.347
epoch:48;    loss:0.348
epoch:49;    loss:0.347
epoch:50;    loss:0.347
Accuracy of the network on the 10000 test images: 64 %
```

- PS：上面print函数忘记改了

样本2

batchnorm+dropout(p=0.5)+2层全连接层+L2正则化+梯度裁剪，valid accuracy=64%

- running_loss=criterion(outputs,labels)

```
epoch:1;     loss:0.409
epoch:2;     loss:0.368
epoch:3;     loss:0.359
epoch:4;     loss:0.357
epoch:5;     loss:0.328
epoch:6;     loss:0.326
epoch:7;     loss:0.321
epoch:8;     loss:0.322
epoch:9;     loss:0.319
epoch:10;     loss:0.299
epoch:11;     loss:0.296
epoch:12;     loss:0.293
epoch:13;     loss:0.291
epoch:14;     loss:0.289
epoch:15;     loss:0.277
epoch:16;     loss:0.274
epoch:17;     loss:0.274
epoch:18;     loss:0.274
epoch:19;     loss:0.273
epoch:20;     loss:0.265
epoch:21;     loss:0.264
epoch:22;     loss:0.264
epoch:23;     loss:0.262
epoch:24;     loss:0.261
epoch:25;     loss:0.257
epoch:26;     loss:0.256
epoch:27;     loss:0.257
epoch:28;     loss:0.255
epoch:29;     loss:0.258
epoch:30;     loss:0.253
epoch:31;     loss:0.253
epoch:32;     loss:0.251
epoch:33;     loss:0.253
epoch:34;     loss:0.252
epoch:35;     loss:0.252
epoch:36;     loss:0.251
epoch:37;     loss:0.251
epoch:38;     loss:0.249
epoch:39;     loss:0.250
epoch:40;     loss:0.249
epoch:41;     loss:0.248
epoch:42;     loss:0.250
epoch:43;     loss:0.251
epoch:44;     loss:0.249
epoch:45;     loss:0.250
epoch:46;     loss:0.249
epoch:47;     loss:0.250
epoch:48;     loss:0.249
epoch:49;     loss:0.248
epoch:50;     loss:0.249
Accuracy of the network on the 10000 valid images: 65 %
root@autodl-container-15da118152-c47fe438:~#
```

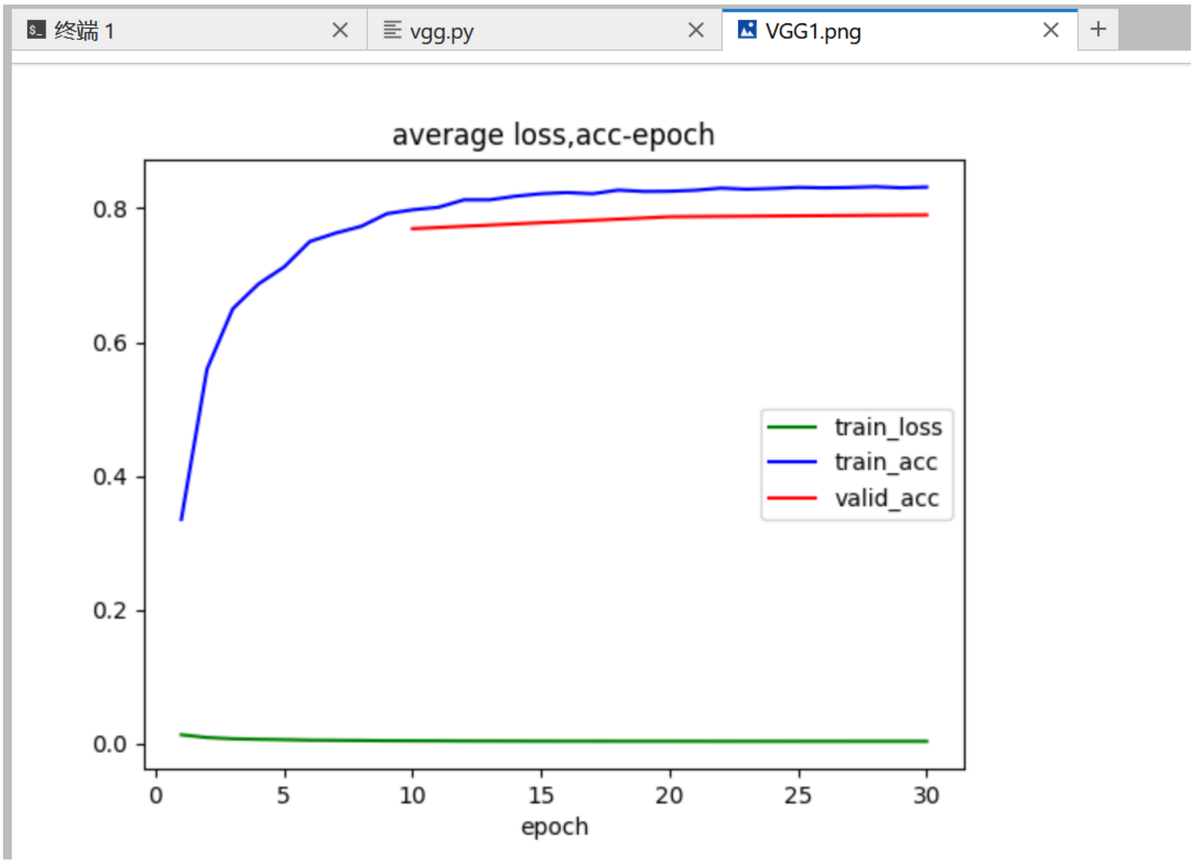
VGG

> 1. 初始
> 2. 作正则化
> 3. 作dropout
> 4. 调超参

order	type
1	全做
2	无
3	正则化
4	dropout
5	图像增强
6	批归一化

vgg1

```
epoch:1;    loss:0.014
epoch:2;    loss:0.010
epoch:3;    loss:0.008
epoch:4;    loss:0.007
epoch:5;    loss:0.006
epoch:6;    loss:0.006
epoch:7;    loss:0.005
epoch:8;    loss:0.005
epoch:9;    loss:0.005
epoch:10;    loss:0.005
Accuracy of the network on the 10000 valid images: 76 %
epoch:11;    loss:0.004
epoch:12;    loss:0.004
epoch:13;    loss:0.004
epoch:14;    loss:0.004
epoch:15;    loss:0.004
epoch:16;    loss:0.004
epoch:17;    loss:0.004
epoch:18;    loss:0.004
epoch:19;    loss:0.004
epoch:20;    loss:0.004
Accuracy of the network on the 10000 valid images: 78 %
epoch:21;    loss:0.004
epoch:22;    loss:0.004
epoch:23;    loss:0.004
epoch:24;    loss:0.004
epoch:25;    loss:0.004
epoch:26;    loss:0.004
epoch:27;    loss:0.004
epoch:28;    loss:0.004
epoch:29;    loss:0.004
epoch:30;    loss:0.004
Accuracy of the network on the 10000 valid images: 79 %
```



vgg2的waterloo

终端 3

```
epoch:8;    loss:0.018
epoch:9;    loss:0.018
epoch:10;    loss:0.018
Accurac of the network on the 10000 valid images: 9 %
epoch:11;    loss:0.018
epoch:12;    loss:0.018
epoch:13;    loss:0.018
epoch:14;    loss:0.018
epoch:15;    loss:0.018
epoch:16;    loss:0.018
epoch:17;    loss:0.018
epoch:18;    loss:0.018
epoch:19;    loss:0.018
epoch:20;    loss:0.018
Accurac of the network on the 10000 valid images: 9 %
epoch:21;    loss:0.018
epoch:22;    loss:0.018
epoch:23;    loss:0.018
epoch:24;    loss:0.018
epoch:25;    loss:0.018
epoch:26;    loss:0.018
epoch:27;    loss:0.018
epoch:28;    loss:0.018
epoch:29;    loss:0.018
epoch:30;    loss:0.018
Accurac of the network on the 10000 valid images: 9 %
root@autodl-container-793811833c-c2928d02:~#
```

后在vgg1中逐一去除下述优化方式，

1. ~~数据增强~~
2. ~~正则化~~
3. 批处理
4. ~~Dropout~~

发现是批处理的影响(其余3个趋势是正确的)

```
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
epoch:1;    loss:0.018
epoch:2;    loss:0.018
epoch:3;    loss:0.018
epoch:4;    loss:0.018
epoch:5;    loss:0.018
epoch:6;    loss:0.018
epoch:7;    loss:0.018
epoch:8;    loss:0.018
epoch:9;    loss:0.018
epoch:10;    loss:0.018
Accuracy of the network on the 10000 valid images: 9 %
epoch:11;    loss:0.018
epoch:12;    loss:0.018
epoch:13;    loss:0.018
epoch:14;    loss:0.018
epoch:15;    loss:0.018
epoch:16;    loss:0.018
epoch:17;    loss:0.018
```

声明

- 由于批处理的影响太大，不进行BN随机性太大，意义不明显，因此组3,4,5均加入批处理理化

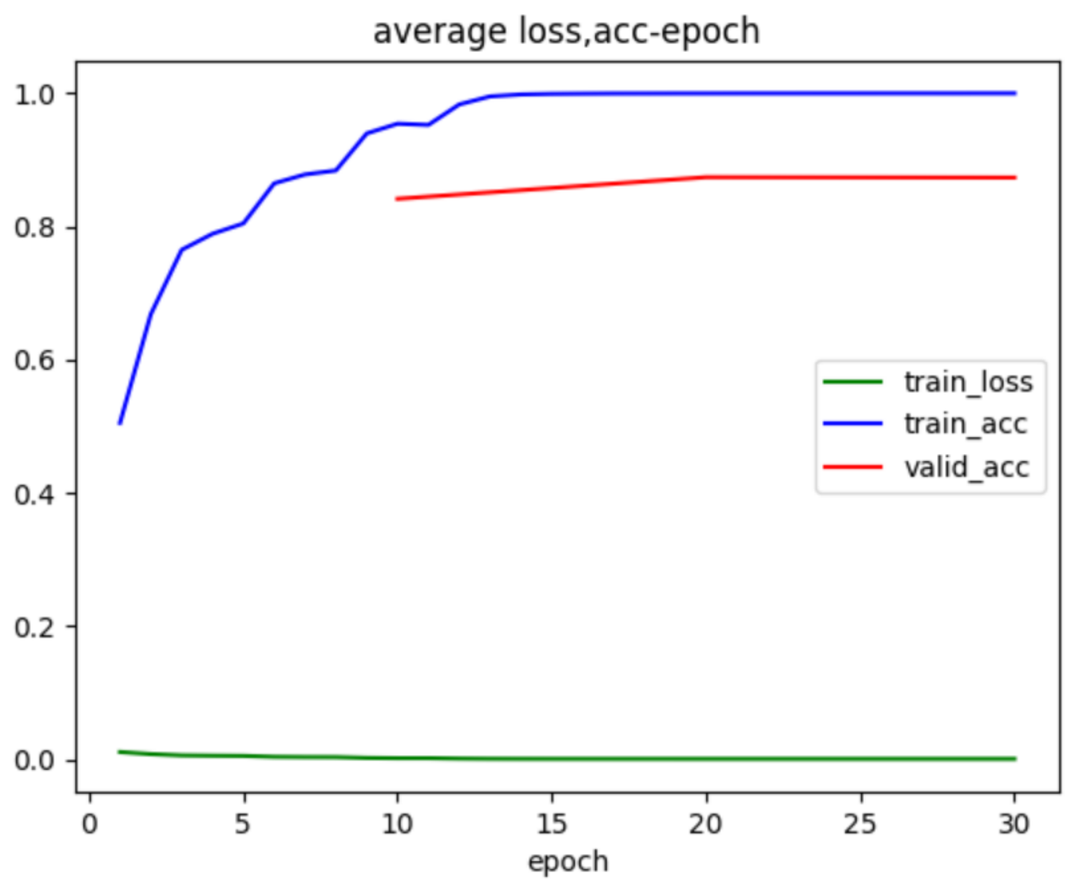
vgg3——正则化+BN的惊人效果

- 在网络较为复杂的时候(VGG13,VGG16,VGG19)，L2正则化项能有效防止过拟合，提升验证准确率!
- 取l2_lambda=0.05

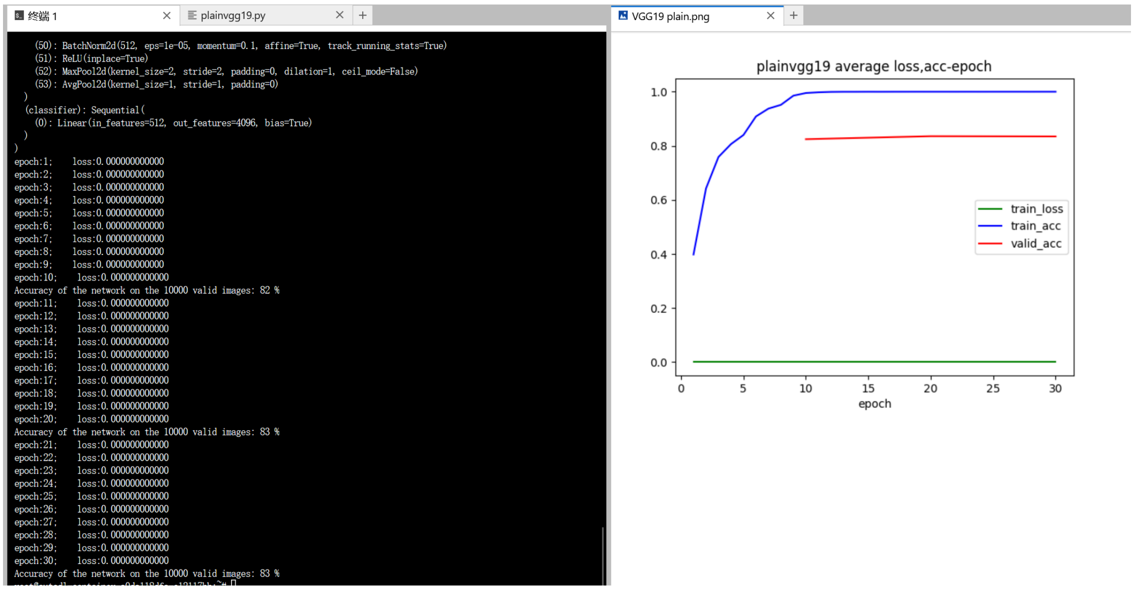
终端 3

```
epoch:8;     loss:0.003
epoch:9;     loss:0.002
epoch:10;     loss:0.001
Accuracy of the network on the 10000 valid images: 84 %
epoch:11;     loss:0.001
epoch:12;     loss:0.001
epoch:13;     loss:0.000
epoch:14;     loss:0.000
epoch:15;     loss:0.000
epoch:16;     loss:0.000
epoch:17;     loss:0.000
epoch:18;     loss:0.000
epoch:19;     loss:0.000
epoch:20;     loss:0.000
Accuracy of the network on the 10000 valid images: 87 %
epoch:21;     loss:0.000
epoch:22;     loss:0.000
epoch:23;     loss:0.000
epoch:24;     loss:0.000
epoch:25;     loss:0.000
epoch:26;     loss:0.000
epoch:27;     loss:0.000
epoch:28;     loss:0.000
epoch:29;     loss:0.000
epoch:30;     loss:0.000
Accuracy of the network on the 10000 valid images: 87 %
root@autodl-container-793811833c-c2928d02:~#
```

VGG3.png 终端 3



- VGG19更为明显，未作过拟合的VGG19

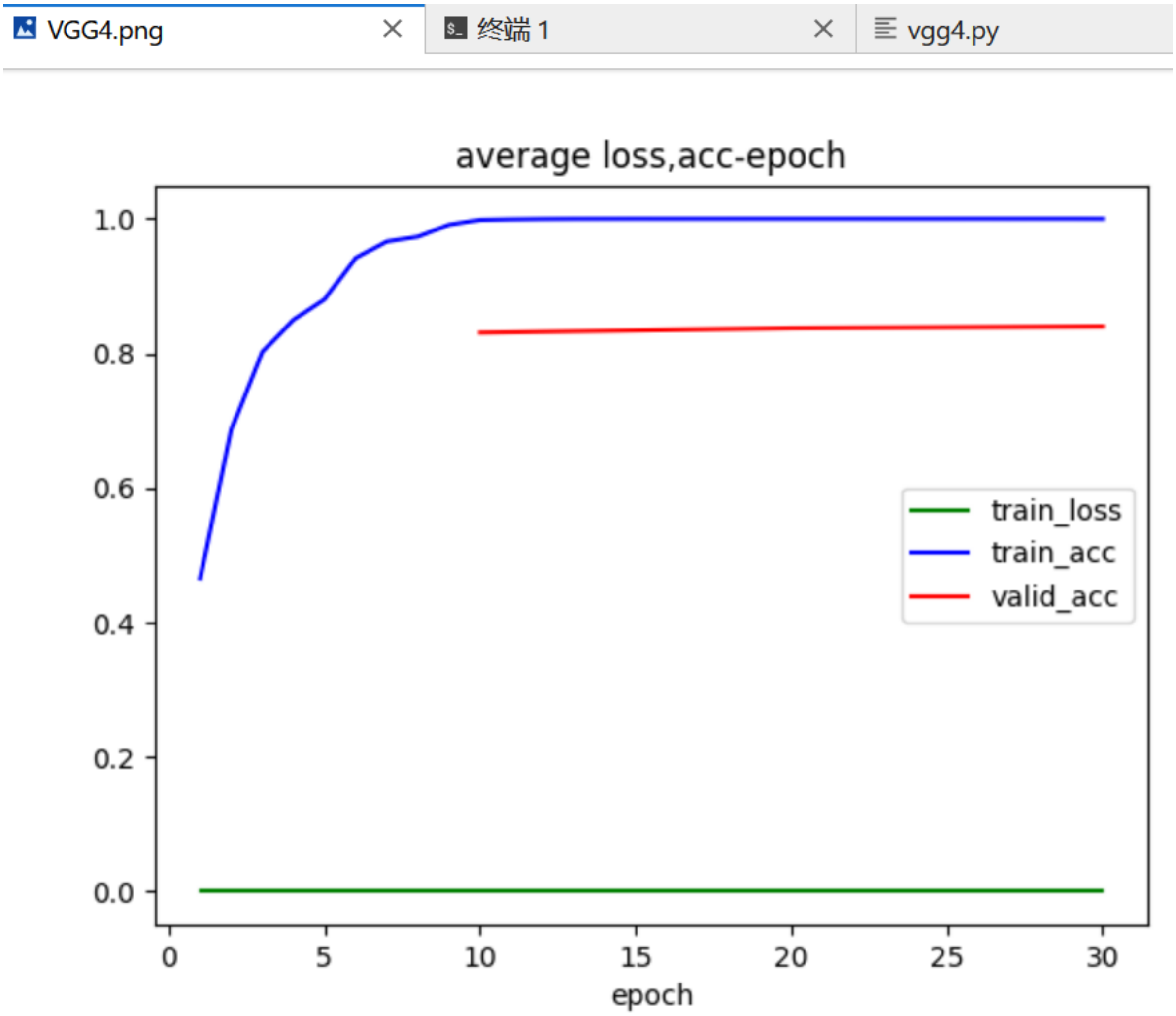


- 过拟合之后的VGG19

vgg4—BN+dropout(0.5)

终端 1 vgg4.py

```
epoch:1;    loss:0.000
epoch:2;    loss:0.000
epoch:3;    loss:0.000
epoch:4;    loss:0.000
epoch:5;    loss:0.000
epoch:6;    loss:0.000
epoch:7;    loss:0.000
epoch:8;    loss:0.000
epoch:9;    loss:0.000
epoch:10;    loss:0.000
Accuracy of the network on the 10000 valid images: 83 %
epoch:11;    loss:0.000
epoch:12;    loss:0.000
epoch:13;    loss:0.000
epoch:14;    loss:0.000
epoch:15;    loss:0.000
epoch:16;    loss:0.000
epoch:17;    loss:0.000
epoch:18;    loss:0.000
epoch:19;    loss:0.000
epoch:20;    loss:0.000
Accuracy of the network on the 10000 valid images: 83 %
epoch:21;    loss:0.000
epoch:22;    loss:0.000
epoch:23;    loss:0.000
epoch:24;    loss:0.000
epoch:25;    loss:0.000
epoch:26;    loss:0.000
epoch:27;    loss:0.000
epoch:28;    loss:0.000
epoch:29;    loss:0.000
epoch:30;    loss:0.000
Accuracy of the network on the 10000 valid images: 84 %
root@autodl-container-28df1197fa-b37alaaa:~# 
```



vggdrop-不同p的drop效果对比

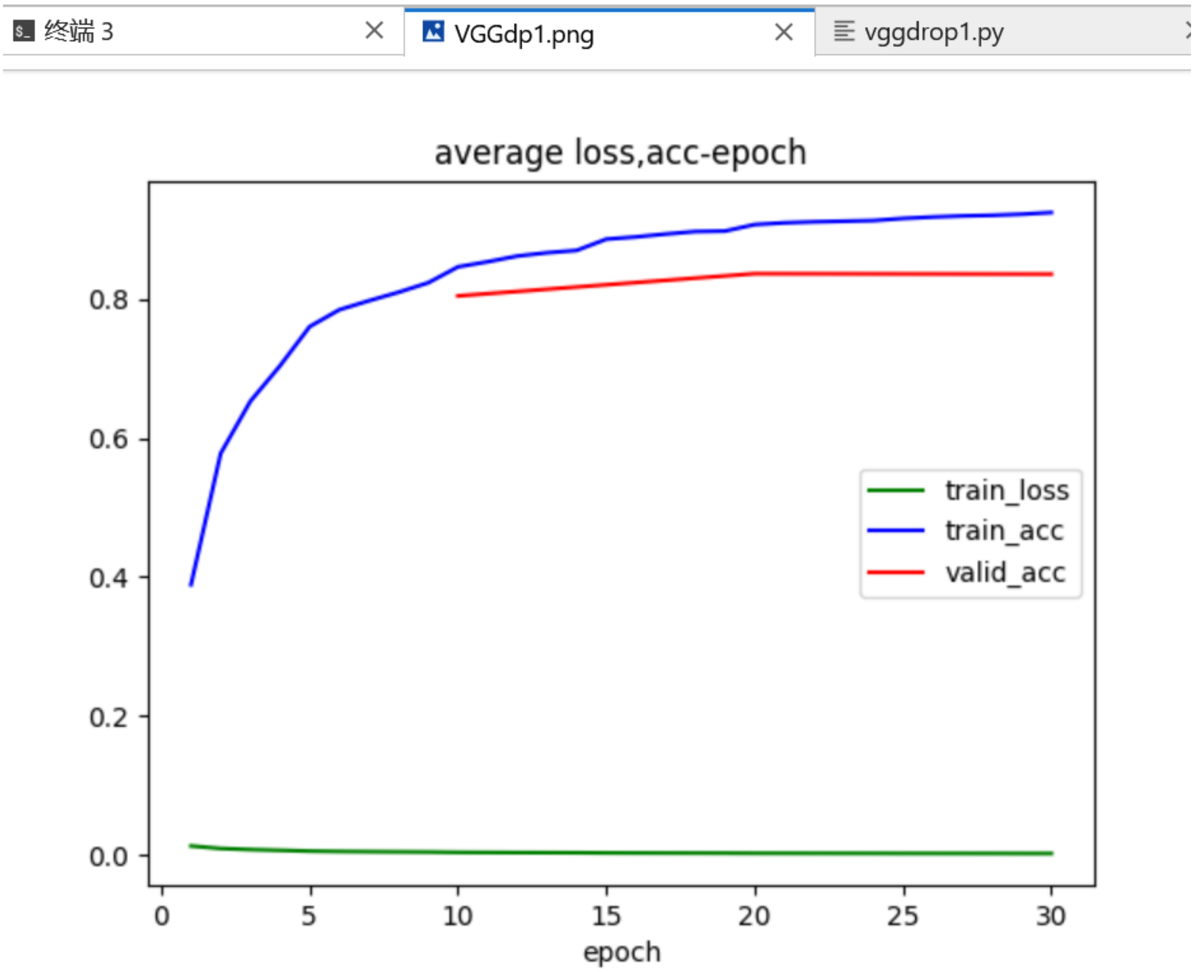
dp	valid_acc(%) with l2_lambda=0.01
0.1	80,83,83
0.2	80,83,84
0.3	81,84,84
0.4	80,83,83
0.5	80,82,83
0.1	(l2_lambda=0.03) 82,85,86
0.1	(l2_lambda=0.05) 80,86,87

- 可见dp太大，反而影响训练效果，在dp=0.3时对训练提升较好

dropout_p=0.1

终端 3　　VGGdp1.png

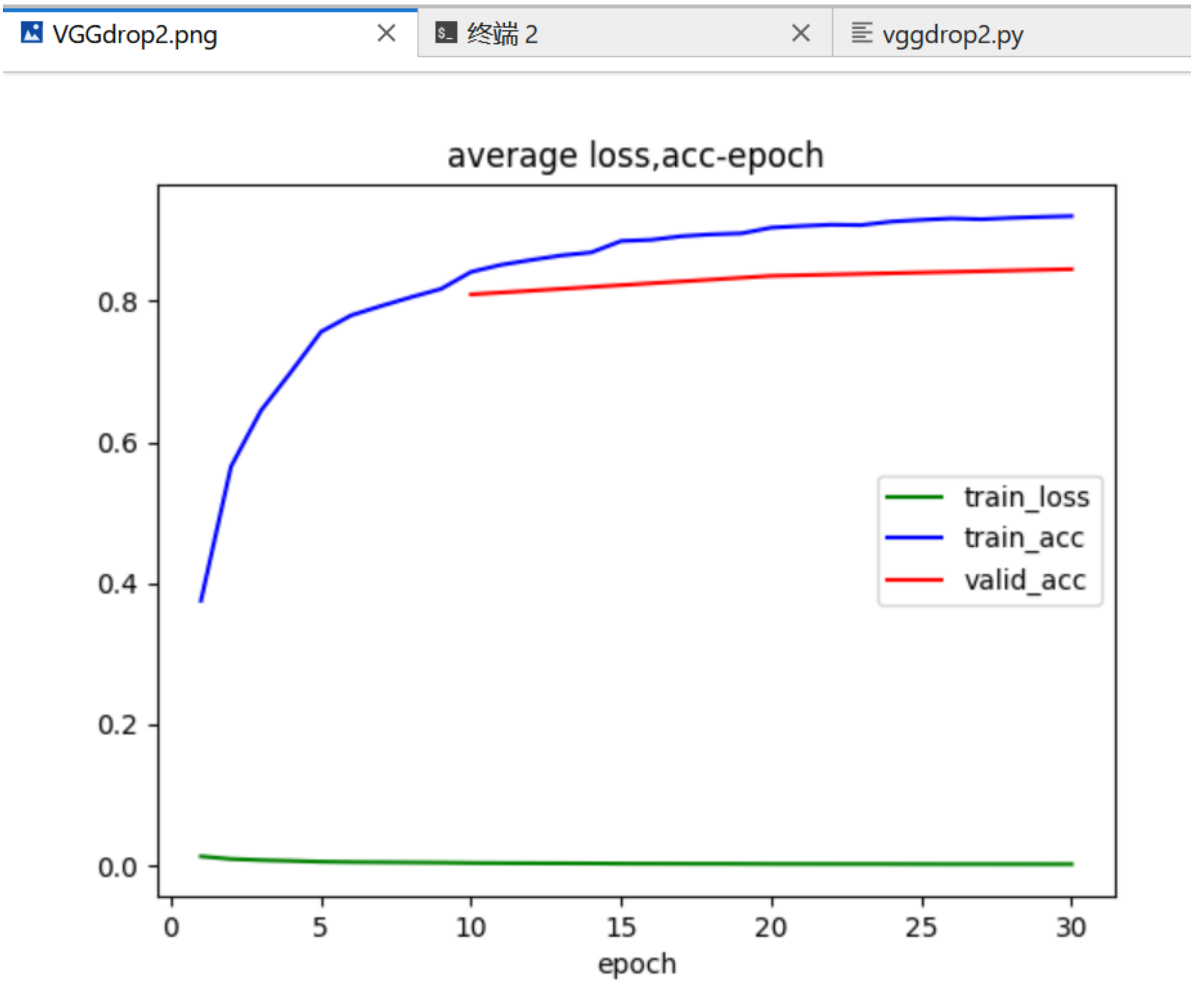
```
epoch:1;     loss:0.013
epoch:2;     loss:0.009
epoch:3;     loss:0.008
epoch:4;     loss:0.007
epoch:5;     loss:0.005
epoch:6;     loss:0.005
epoch:7;     loss:0.005
epoch:8;     loss:0.004
epoch:9;     loss:0.004
epoch:10;     loss:0.004
Accuracy of the network on the 10000 valid images: 80 %
epoch:11;     loss:0.003
epoch:12;     loss:0.003
epoch:13;     loss:0.003
epoch:14;     loss:0.003
epoch:15;     loss:0.003
epoch:16;     loss:0.003
epoch:17;     loss:0.003
epoch:18;     loss:0.002
epoch:19;     loss:0.002
epoch:20;     loss:0.002
Accuracy of the network on the 10000 valid images: 83 %
epoch:21;     loss:0.002
epoch:22;     loss:0.002
epoch:23;     loss:0.002
epoch:24;     loss:0.002
epoch:25;     loss:0.002
epoch:26;     loss:0.002
epoch:27;     loss:0.002
epoch:28;     loss:0.002
epoch:29;     loss:0.002
epoch:30;     loss:0.002
Accuracy of the network on the 10000 valid images: 83 %
root@autodl-container-793811833c-c2928d02:~#
```

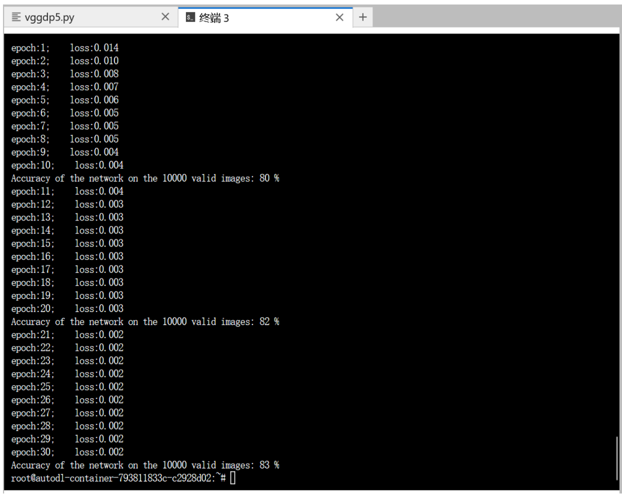


- 可见过拟合了

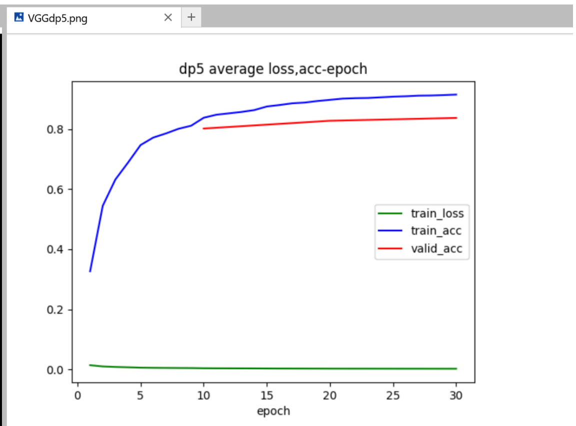
dropout_p=0.2

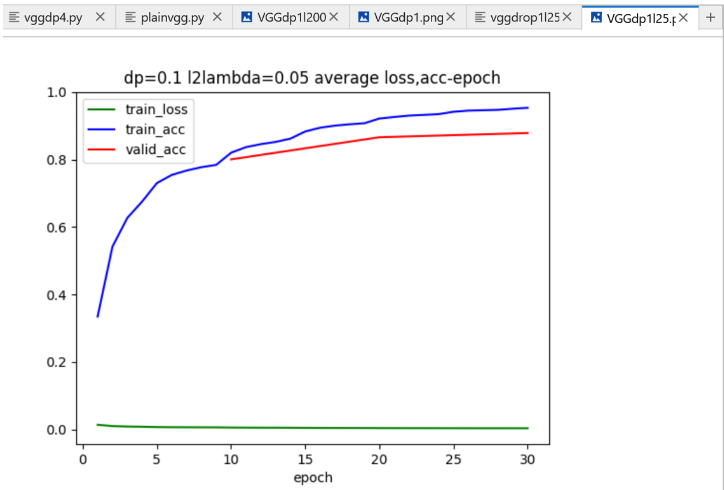
```
epoch:1;    loss:0.013
epoch:2;    loss:0.009
epoch:3;    loss:0.008
epoch:4;    loss:0.007
epoch:5;    loss:0.005
epoch:6;    loss:0.005
epoch:7;    loss:0.005
epoch:8;    loss:0.004
epoch:9;    loss:0.004
epoch:10;    loss:0.004
Accuracy of the network on the 10000 valid images: 80 %
epoch:11;    loss:0.003
epoch:12;    loss:0.003
epoch:13;    loss:0.003
epoch:14;    loss:0.003
epoch:15;    loss:0.003
epoch:16;    loss:0.003
epoch:17;    loss:0.003
epoch:18;    loss:0.003
epoch:19;    loss:0.002
epoch:20;    loss:0.002
Accuracy of the network on the 10000 valid images: 83 %
epoch:21;    loss:0.002
epoch:22;    loss:0.002
epoch:23;    loss:0.002
epoch:24;    loss:0.002
epoch:25;    loss:0.002
epoch:26;    loss:0.002
epoch:27;    loss:0.002
epoch:28;    loss:0.002
epoch:29;    loss:0.002
epoch:30;    loss:0.002
Accuracy of the network on the 10000 valid images: 84 %
```

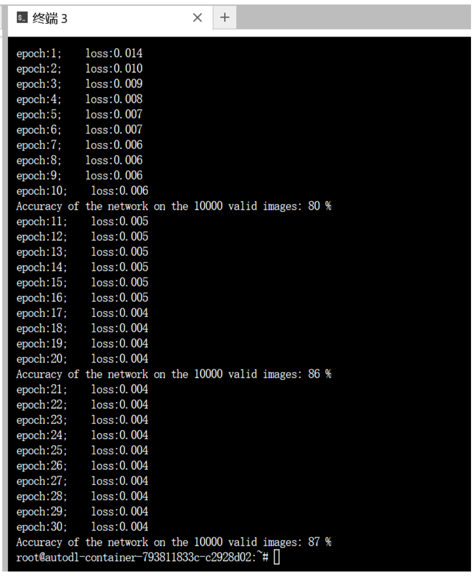



vggdp5.py 终端 3

```
epoch:1;    loss:0.014
epoch:2;    loss:0.010
epoch:3;    loss:0.008
epoch:4;    loss:0.007
epoch:5;    loss:0.006
epoch:6;    loss:0.005
epoch:7;    loss:0.005
epoch:8;    loss:0.005
epoch:9;    loss:0.004
epoch:10;    loss:0.004
Accuracy of the network on the 10000 valid images: 80 %
epoch:11;    loss:0.004
epoch:12;    loss:0.003
epoch:13;    loss:0.003
epoch:14;    loss:0.003
epoch:15;    loss:0.003
epoch:16;    loss:0.003
epoch:17;    loss:0.003
epoch:18;    loss:0.003
epoch:19;    loss:0.003
epoch:20;    loss:0.003
Accuracy of the network on the 10000 valid images: 82 %
epoch:21;    loss:0.002
epoch:22;    loss:0.002
epoch:23;    loss:0.002
epoch:24;    loss:0.002
epoch:25;    loss:0.002
epoch:26;    loss:0.002
epoch:27;    loss:0.002
epoch:28;    loss:0.002
epoch:29;    loss:0.002
epoch:30;    loss:0.002
Accuracy of the network on the 10000 valid images: 83 %
root@autodl-container-793811833c-c2928d02:~#
```





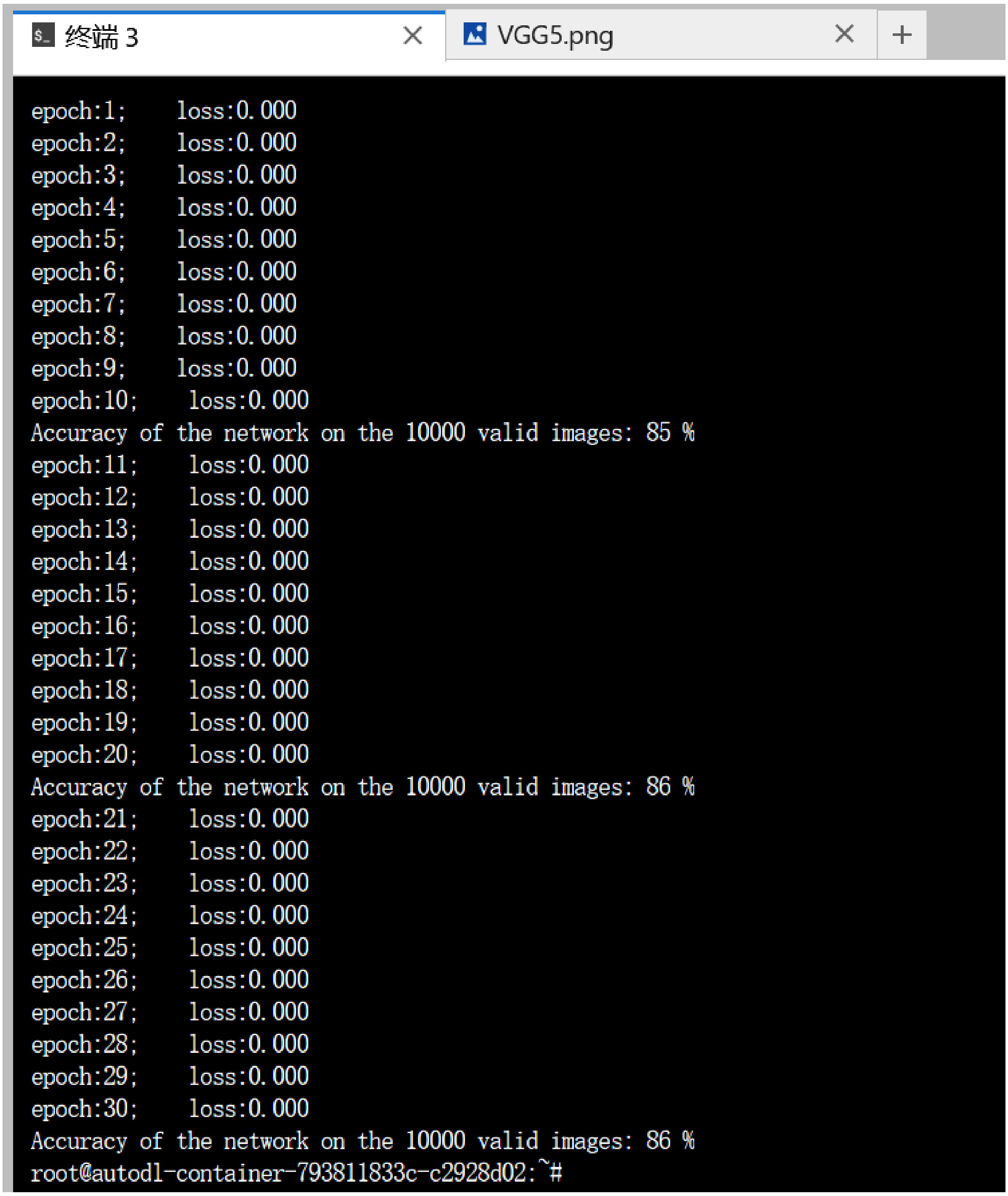

终端 3

```
epoch:1;    loss:0.014
epoch:2;    loss:0.010
epoch:3;    loss:0.009
epoch:4;    loss:0.008
epoch:5;    loss:0.007
epoch:6;    loss:0.007
epoch:7;    loss:0.006
epoch:8;    loss:0.006
epoch:9;    loss:0.006
epoch:10;    loss:0.006
Accuracy of the network on the 10000 valid images: 80 %
epoch:11;    loss:0.005
epoch:12;    loss:0.005
epoch:13;    loss:0.005
epoch:14;    loss:0.005
epoch:15;    loss:0.005
epoch:16;    loss:0.005
epoch:17;    loss:0.004
epoch:18;    loss:0.004
epoch:19;    loss:0.004
epoch:20;    loss:0.004
Accuracy of the network on the 10000 valid images: 86 %
epoch:21;    loss:0.004
epoch:22;    loss:0.004
epoch:23;    loss:0.004
epoch:24;    loss:0.004
epoch:25;    loss:0.004
epoch:26;    loss:0.004
epoch:27;    loss:0.004
epoch:28;    loss:0.004
epoch:29;    loss:0.004
epoch:30;    loss:0.004
Accuracy of the network on the 10000 valid images: 87 %
root@autodl-container-793811833c-c2928d02:~#
```

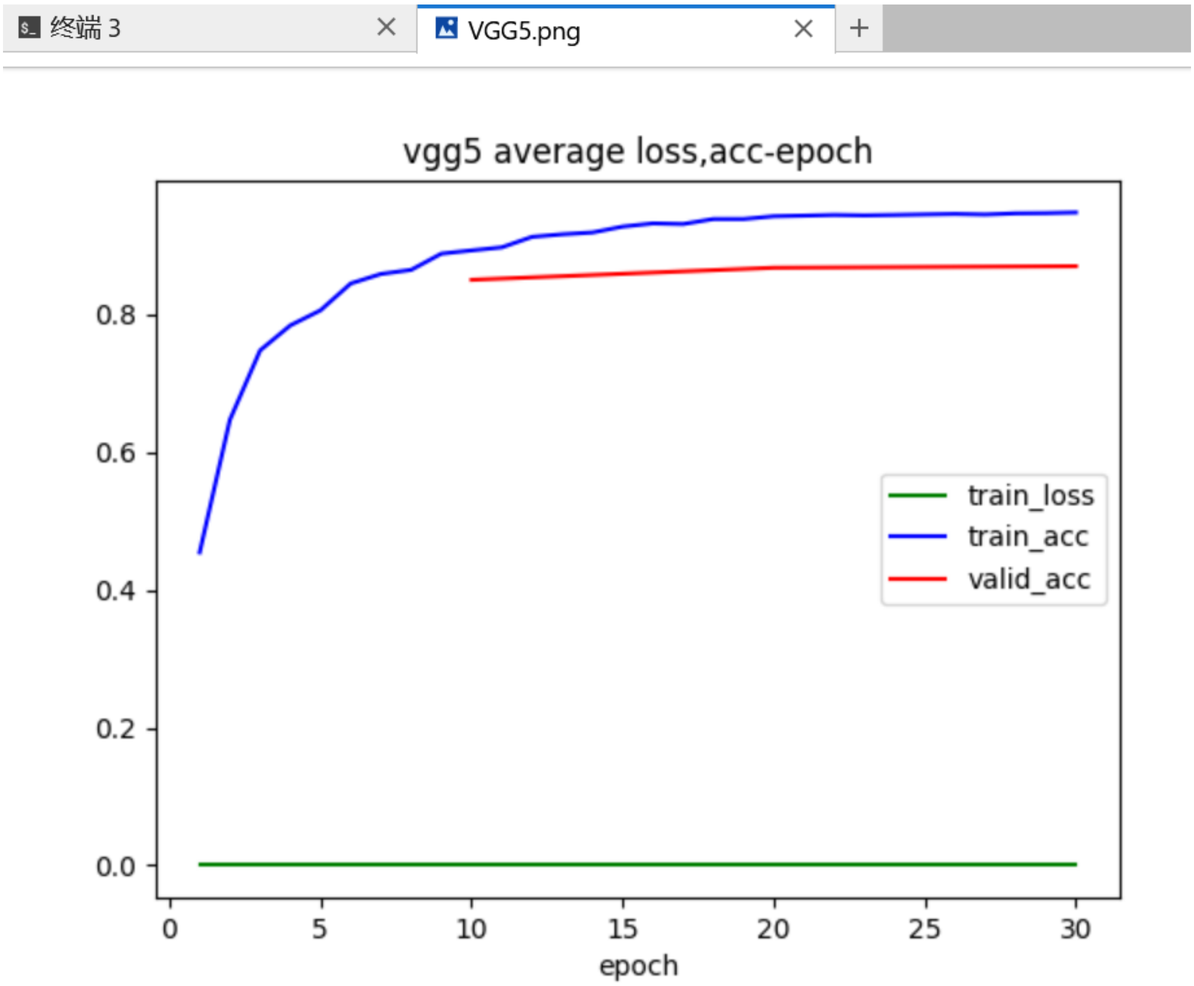
- dropout在xxx比较好

vgg5-收敛最快的BN+pic_intensify

- 可见图像增强可显著提升训练效果

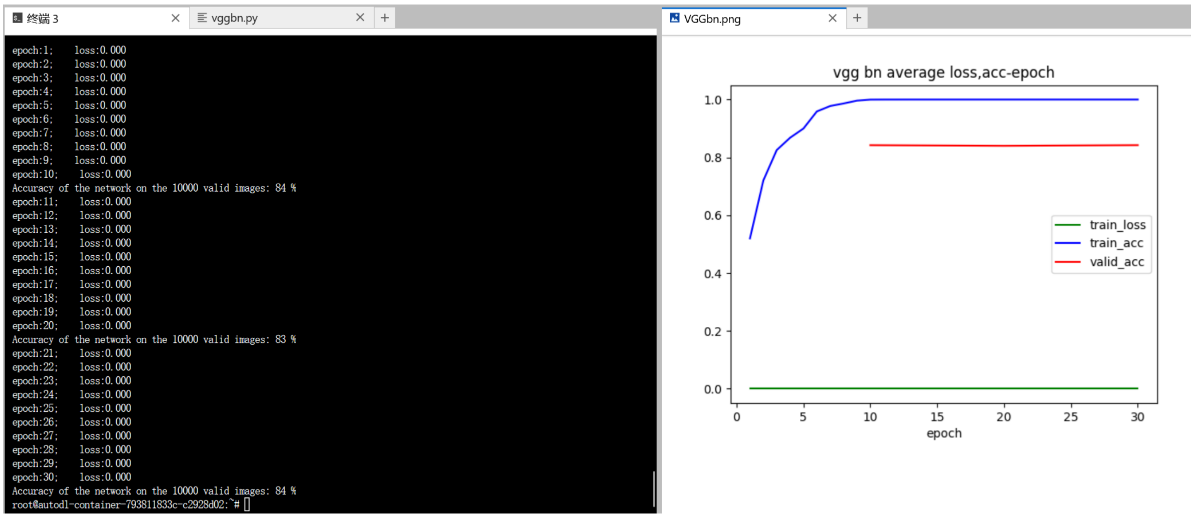



```
epoch:1;    loss:0.000
epoch:2;    loss:0.000
epoch:3;    loss:0.000
epoch:4;    loss:0.000
epoch:5;    loss:0.000
epoch:6;    loss:0.000
epoch:7;    loss:0.000
epoch:8;    loss:0.000
epoch:9;    loss:0.000
epoch:10;    loss:0.000
Accuracy of the network on the 10000 valid images: 85 %
epoch:11;    loss:0.000
epoch:12;    loss:0.000
epoch:13;    loss:0.000
epoch:14;    loss:0.000
epoch:15;    loss:0.000
epoch:16;    loss:0.000
epoch:17;    loss:0.000
epoch:18;    loss:0.000
epoch:19;    loss:0.000
epoch:20;    loss:0.000
Accuracy of the network on the 10000 valid images: 86 %
epoch:21;    loss:0.000
epoch:22;    loss:0.000
epoch:23;    loss:0.000
epoch:24;    loss:0.000
epoch:25;    loss:0.000
epoch:26;    loss:0.000
epoch:27;    loss:0.000
epoch:28;    loss:0.000
epoch:29;    loss:0.000
epoch:30;    loss:0.000
Accuracy of the network on the 10000 valid images: 86 %
root@autodl-container-793811833c-c2928d02:~#
```



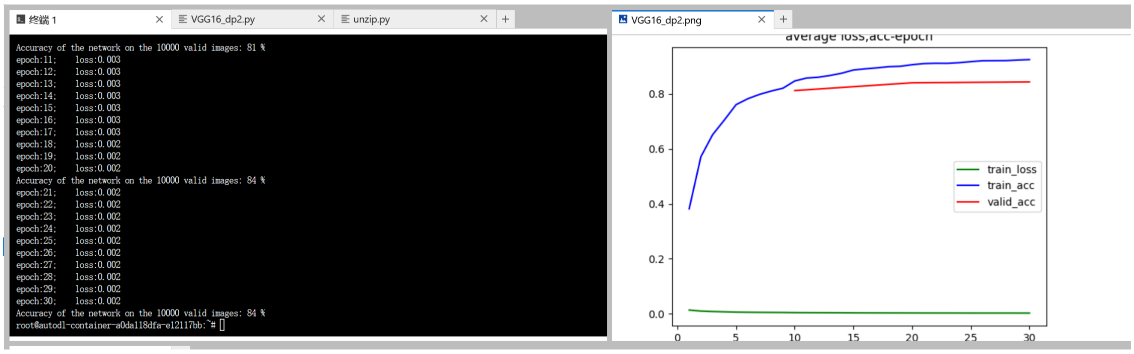
vgg6—单BN，cnn的顶梁柱

- only-bn将训练效果从vgg2的9%感人正确率提高到84%，其他的只是锦上添花~

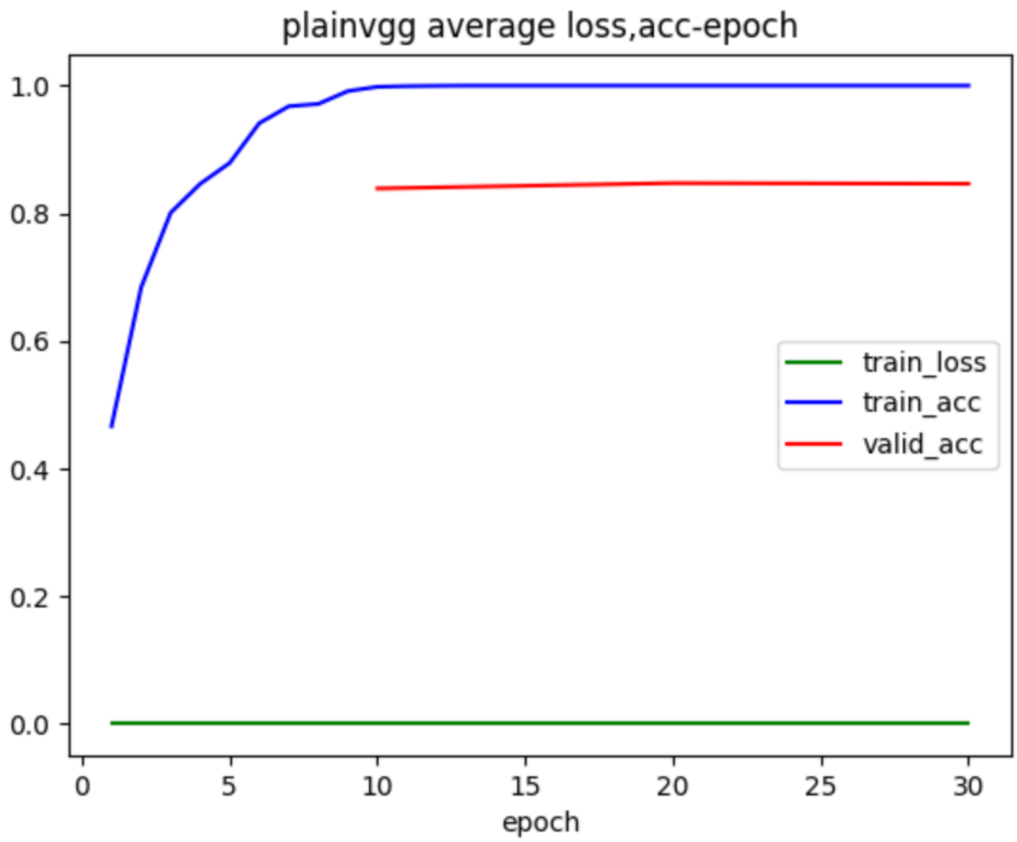


VGG16/19

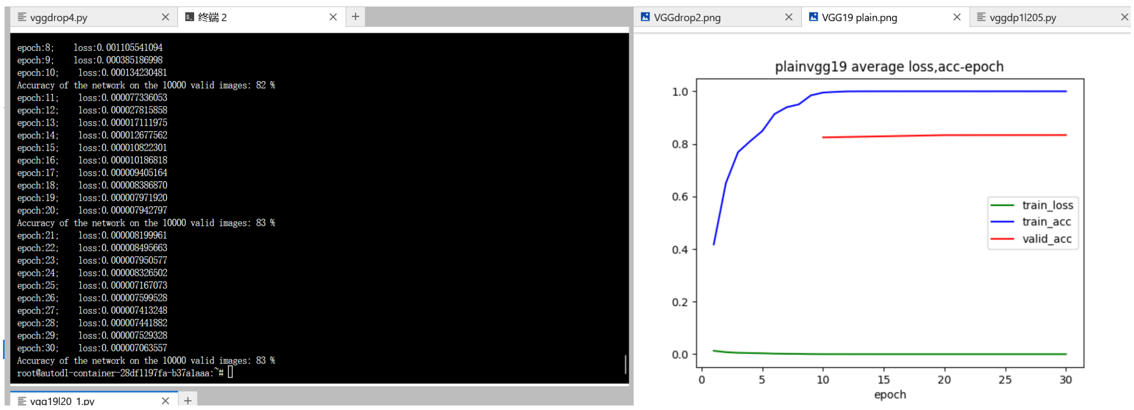
- 由于前面BN后，l2 regulation和pic_intensify取得了最好的效果，
- vgg16+vgg1的配置(全配置)+dp=0.2，效果并不好



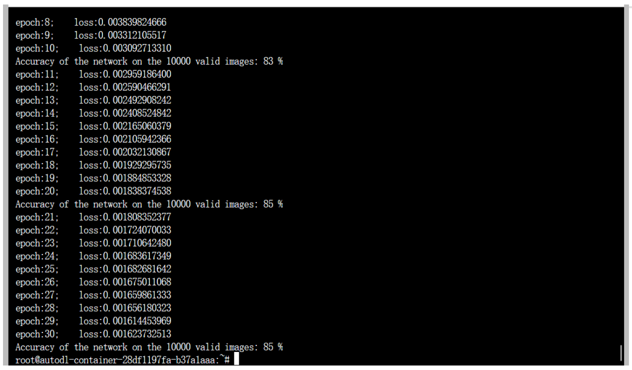
- 试了下只有BN的配置，dp=0.5的配置，效果依然不好，83，83，84

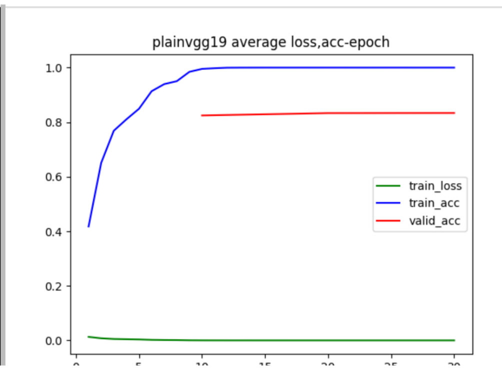


l2=0.1,82，83，83；l2=0.05 完全一样，



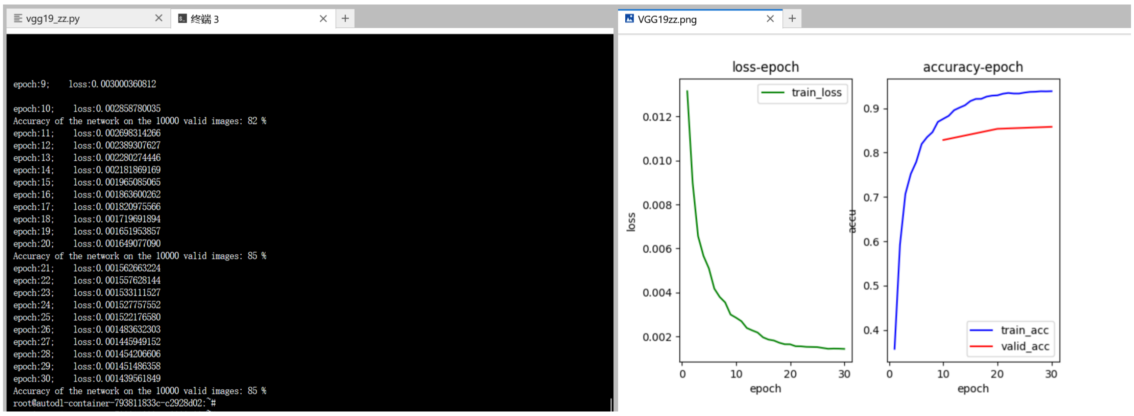
- 可能欠拟合，l2=0.01+数据增强，效果如下



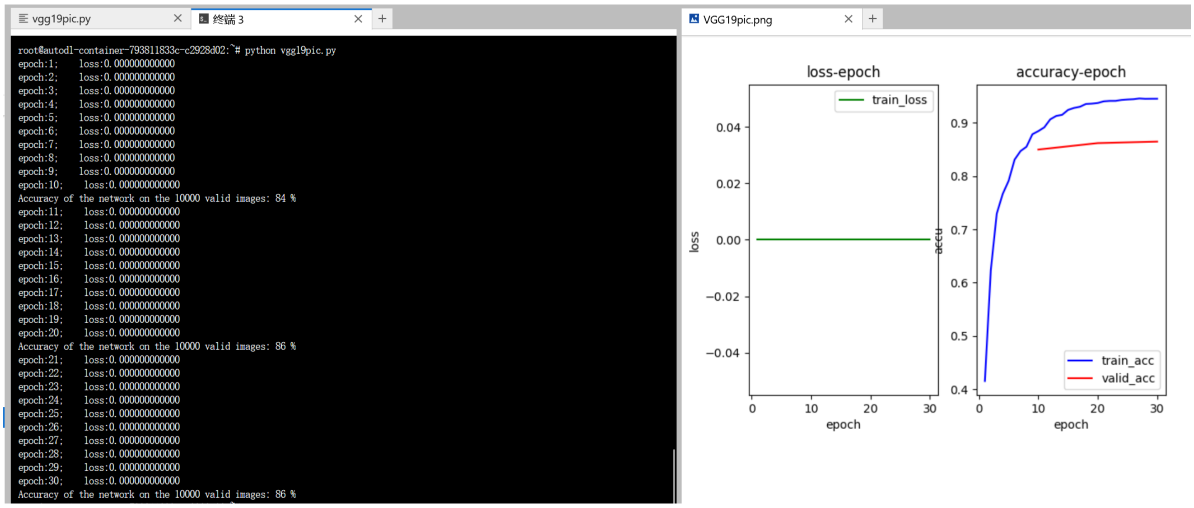


- l2=0.05+图像增强，82，84，85
- 采用l2_λ=0.05+pic,卷积池化之后灵活的3层输出并dropout，效果如下:

dp's p	effect	
0(vgg19yy)	83,86,86	
0.3(vgg19xx)	83,86,86	
0.5(vgg19zz)	83,85,85	

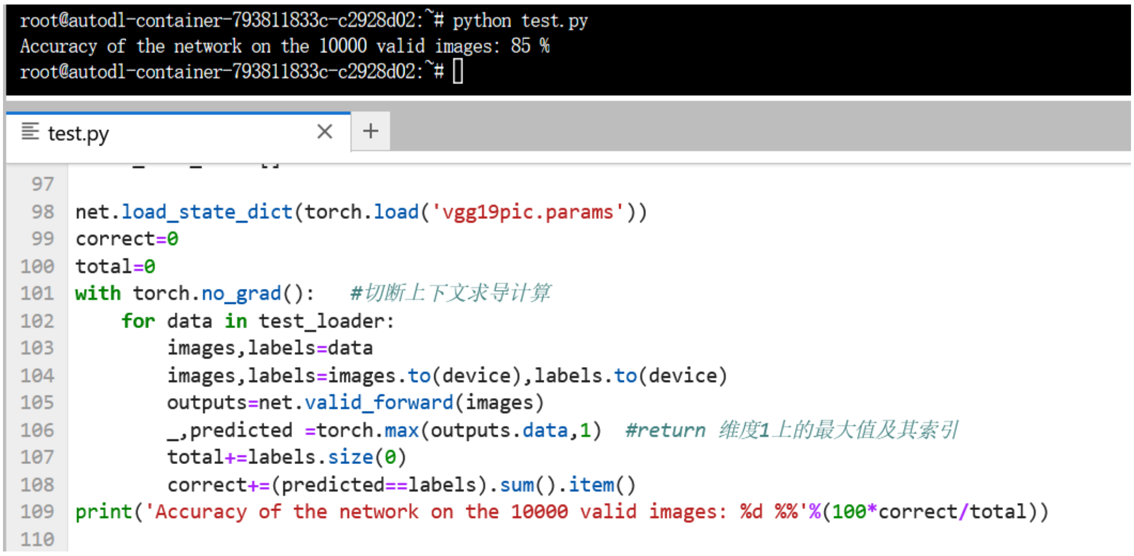


- vgg19+bn+l2regulation,84,85,86
- vgg19+bn+pic_intensify,84,86,86



test

- vgg19+bn+pic_intensify验证效果最优，我们就以它为测试组，给出我们的答案85%

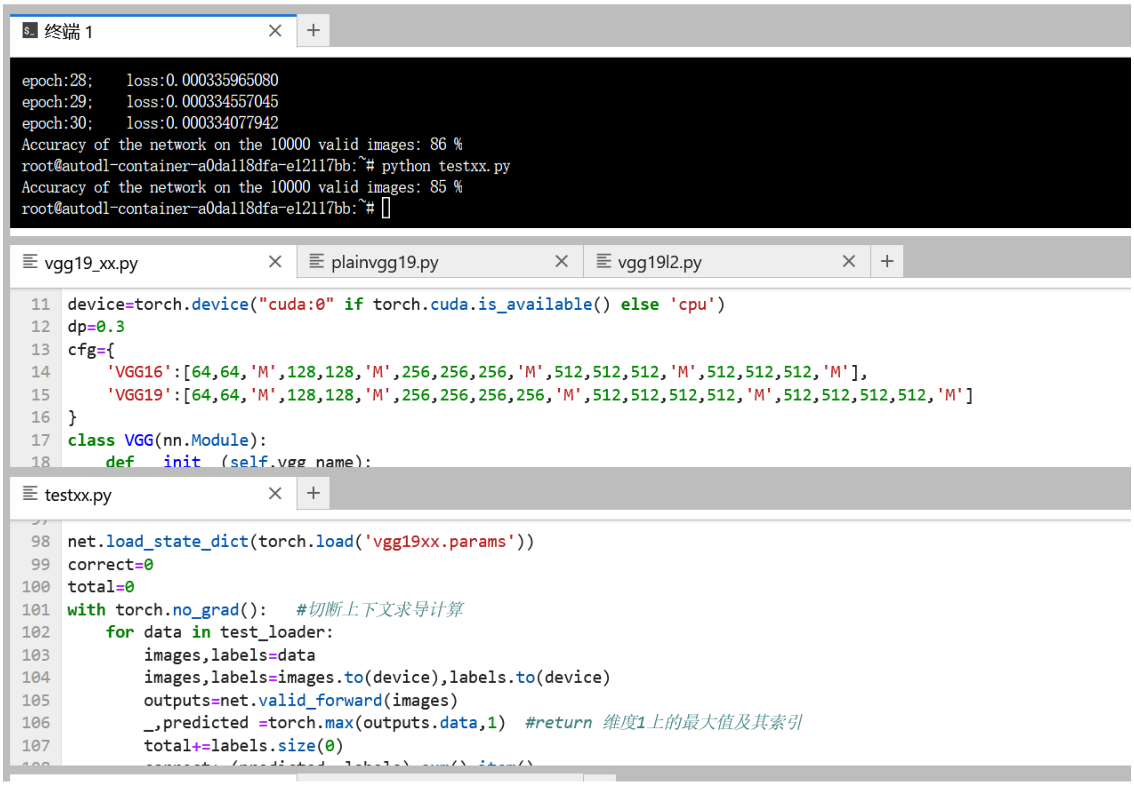


```
root@autodl-container-793811833c-c2928d02:~# python test.py
Accuracy of the network on the 10000 valid images: 85 %
root@autodl-container-793811833c-c2928d02:~#
```

test.py

```
net.load_state_dict(torch.load('vgg19pic.params'))
correct=0
total=0
with torch.no_grad():   #切断上下文求导计算
    for data in test_loader:
        images,labels=data
        images,labels=images.to(device),labels.to(device)
        outputs=net.valid_forward(images)
        _,predicted =torch.max(outputs.data,1)  #return 维度1上的最大值及其索引
        total+=labels.size(0)
        correct+=(predicted==labels).sum().item()
print('Accuracy of the network on the 10000 valid images: %d %%'%(100*correct/total))
```

- 为了体现用到了l2_regulation和dropout，还用vgg19_xx试了以下(前面的+dp=0.3,l2_lambda=0.1)，效果也是85%，~~其实vggyy也是85%~~



终端 1

```
epoch:28;    loss:0.000335965080
epoch:29;    loss:0.000334557045
epoch:30;    loss:0.000334077942
Accuracy of the network on the 10000 valid images: 86 %
root@autodl-container-a0da118dfa-e12117bb:~# python testxx.py
Accuracy of the network on the 10000 valid images: 85 %
root@autodl-container-a0da118dfa-e12117bb:~#
```

vgg19_xx.py　plainvgg19.py　vgg19l2.py

```
device=torch.device("cuda:0" if torch.cuda.is_available() else 'cpu')
dp=0.3
cfg={
    'VGG16':[64,64,'M',128,128,'M',256,256,256,'M',512,512,512,'M',512,512,512,'M'],
    'VGG19':[64,64,'M',128,128,'M',256,256,256,256,'M',512,512,512,512,'M',512,512,512,512,'M']
}
class VGG(nn.Module):
    def __init__(self,vgg_name):
```

testxx.py

```
net.load_state_dict(torch.load('vgg19xx.params'))
correct=0
total=0
with torch.no_grad():   #切断上下文求导计算
    for data in test_loader:
        images,labels=data
        images,labels=images.to(device),labels.to(device)
        outputs=net.valid_forward(images)
        _,predicted =torch.max(outputs.data,1)  #return 维度1上的最大值及其索引
        total+=labels.size(0)
```

summary

1. 大部分(尤其开始的时候)，训练速度比训练精度更重要，因为可以及时获得反馈
2. unzip(上传zip到云GPUserver后)

```
import zipfile
import os

# 压缩文件路径
zip_path='data.zip'

# 文件存储路径
```

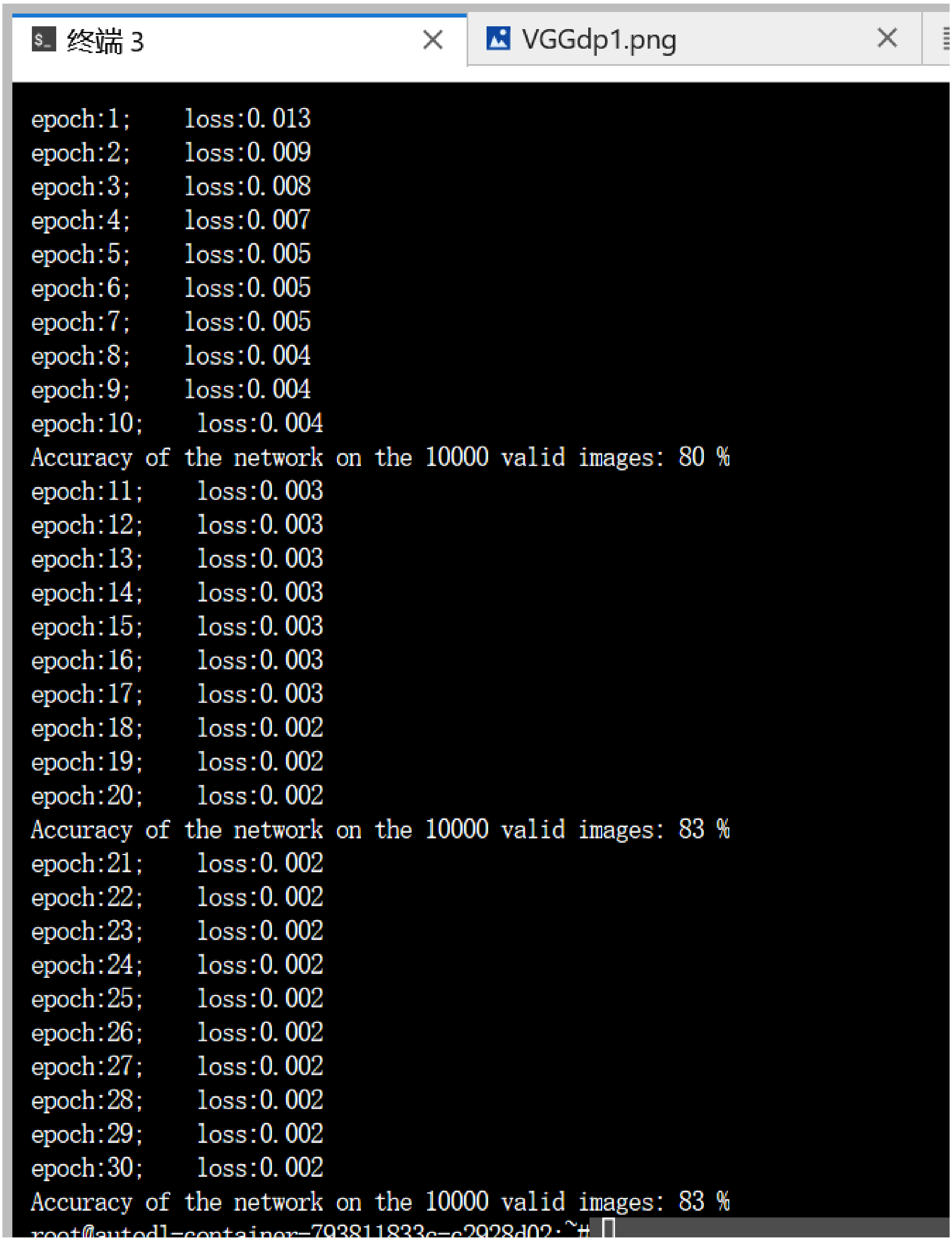
```python
save_path = '.'

# 读取压缩文件
file=zipfile.ZipFile(zip_path)

# 解压文件
print('开始解压...')
file.extractall(save_path)
print('解压结束。')
```

3. 在简单网络的训练中发现一个很有意思的现象，在$\equiv 4(mod 5) \to \equiv 5(mod 5)$，会出现相比其他周期较大幅度的降低，这恰好是学习率衰减(lr*=0.5)的时候

```
epoch:1;    loss:0.633
epoch:2;    loss:0.474
epoch:3;    loss:0.468
epoch:4;    loss:0.465
epoch:5;    loss:0.431
epoch:6;    loss:0.424
epoch:7;    loss:0.422
epoch:8;    loss:0.420
epoch:9;    loss:0.419
epoch:10;    loss:0.399
epoch:11;    loss:0.396
epoch:12;    loss:0.392
epoch:13;    loss:0.393
epoch:14;    loss:0.391
epoch:15;    loss:0.378
epoch:16;    loss:0.376
epoch:17;    loss:0.375
epoch:18;    loss:0.373
epoch:19;    loss:0.372
epoch:20;    loss:0.365
epoch:21;    loss:0.362
epoch:22;    loss:0.362
epoch:23;    loss:0.362
epoch:24;    loss:0.362
epoch:25;    loss:0.356
epoch:26;    loss:0.357
epoch:27;    loss:0.357
epoch:28;    loss:0.357
epoch:29;    loss:0.355
epoch:30;    loss:0.352
epoch:31;    loss:0.352
epoch:32;    loss:0.352
epoch:33;    loss:0.352
```

4. 在看到test和valid相差较大后，可适当调大l2_lambda，防止过拟合，

在VGG4中(BN+dropout)，同样是dropout's p=0.1,将l2_lambda从0.01调为0.03，测试精度确有提升



终端 3 VGGdp1.png

```
epoch:1;     loss:0.013
epoch:2;     loss:0.009
epoch:3;     loss:0.008
epoch:4;     loss:0.007
epoch:5;     loss:0.005
epoch:6;     loss:0.005
epoch:7;     loss:0.005
epoch:8;     loss:0.004
epoch:9;     loss:0.004
epoch:10;     loss:0.004
Accuracy of the network on the 10000 valid images: 80 %
epoch:11;     loss:0.003
epoch:12;     loss:0.003
epoch:13;     loss:0.003
epoch:14;     loss:0.003
epoch:15;     loss:0.003
epoch:16;     loss:0.003
epoch:17;     loss:0.003
epoch:18;     loss:0.002
epoch:19;     loss:0.002
epoch:20;     loss:0.002
Accuracy of the network on the 10000 valid images: 83 %
epoch:21;     loss:0.002
epoch:22;     loss:0.002
epoch:23;     loss:0.002
epoch:24;     loss:0.002
epoch:25;     loss:0.002
epoch:26;     loss:0.002
epoch:27;     loss:0.002
epoch:28;     loss:0.002
epoch:29;     loss:0.002
epoch:30;     loss:0.002
Accuracy of the network on the 10000 valid images: 83 %
root@autodl-container-793811833c-c2928d02:~#
```

终端 3　　VGGdp1l2003.png

```
epoch:1;    loss:0.013
epoch:2;    loss:0.009
epoch:3;    loss:0.008
epoch:4;    loss:0.007
epoch:5;    loss:0.006
epoch:6;    loss:0.005
epoch:7;    loss:0.005
epoch:8;    loss:0.005
epoch:9;    loss:0.005
epoch:10;    loss:0.004
Accuracy of the network on the 10000 valid images: 82 %
epoch:11;    loss:0.004
epoch:12;    loss:0.004
epoch:13;    loss:0.003
epoch:14;    loss:0.003
epoch:15;    loss:0.003
epoch:16;    loss:0.003
epoch:17;    loss:0.003
epoch:18;    loss:0.003
epoch:19;    loss:0.003
epoch:20;    loss:0.002
Accuracy of the network on the 10000 valid images: 85 %
epoch:21;    loss:0.002
epoch:22;    loss:0.002
epoch:23;    loss:0.002
epoch:24;    loss:0.002
epoch:25;    loss:0.002
epoch:26;    loss:0.002
epoch:27;    loss:0.002
epoch:28;    loss:0.002
epoch:29;    loss:0.002
epoch:30;    loss:0.002
Accuracy of the network on the 10000 valid images: 86 %
```

4. 用一个pic_density就能取得很好的效果，再用l2regulation或dropout效果反而稍降~~
5. 下次可以尝试下更灵活的学习率调整，

   scheduler=optim.lr_scheduler.stepLR(optimizer,step_size=10,gamma=0.01)

```python
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

# 在训练时更新学习率
for epoch in range(num_epochs):
    # ...

    # 更新optimizer中的学习率
    scheduler.step()
```

6. 如何把训练结果保存输出?

```python
torch.save(net.state_dict(),'mlp.params')
clone=MLP()
clone.load_state_dict(torch.load('mlp.params'))
clone.eval()
```

7. 初级训练方式为设置对照组，每次更改一个变量观察其影响，再组合那些较好的变量setting(当然可能正正得负，防过拟合过度→欠拟合)，有点像生化幻彩的炼丹，大概是自己理论基础不好
8. 要调的参数比较多，经常丢三落四，挺费时间的
9. 准确率和损失相差较大，用2个图显示比较好

   ```python
   s=plt.figure()
   b=s.add_subplot(1,2,1)
   c=s.add_subplot(1,2,2)
   #子图和普通图一样处理即可,标签要加'set_'
   ```

10. 还有googlenet(1*1卷积)，Res网络(卷积层F(x)拟合h(x)-x)没有实践，我们的VGG13用最大池化层，VGG16/19则先用最大池化处理卷积结果，最后附加一个平均池化层，和NiN类似