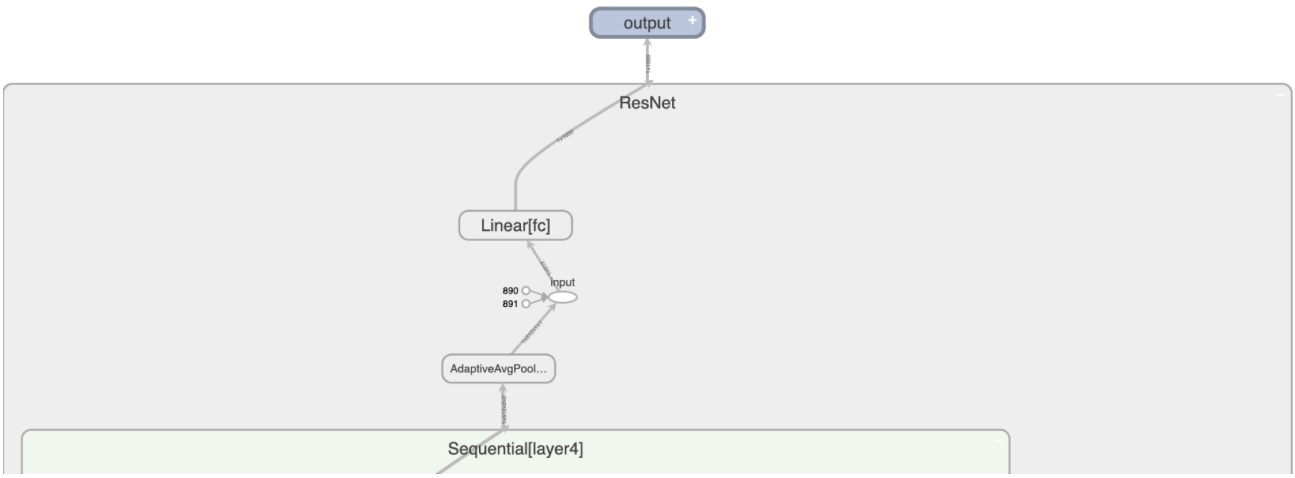
DL HW2

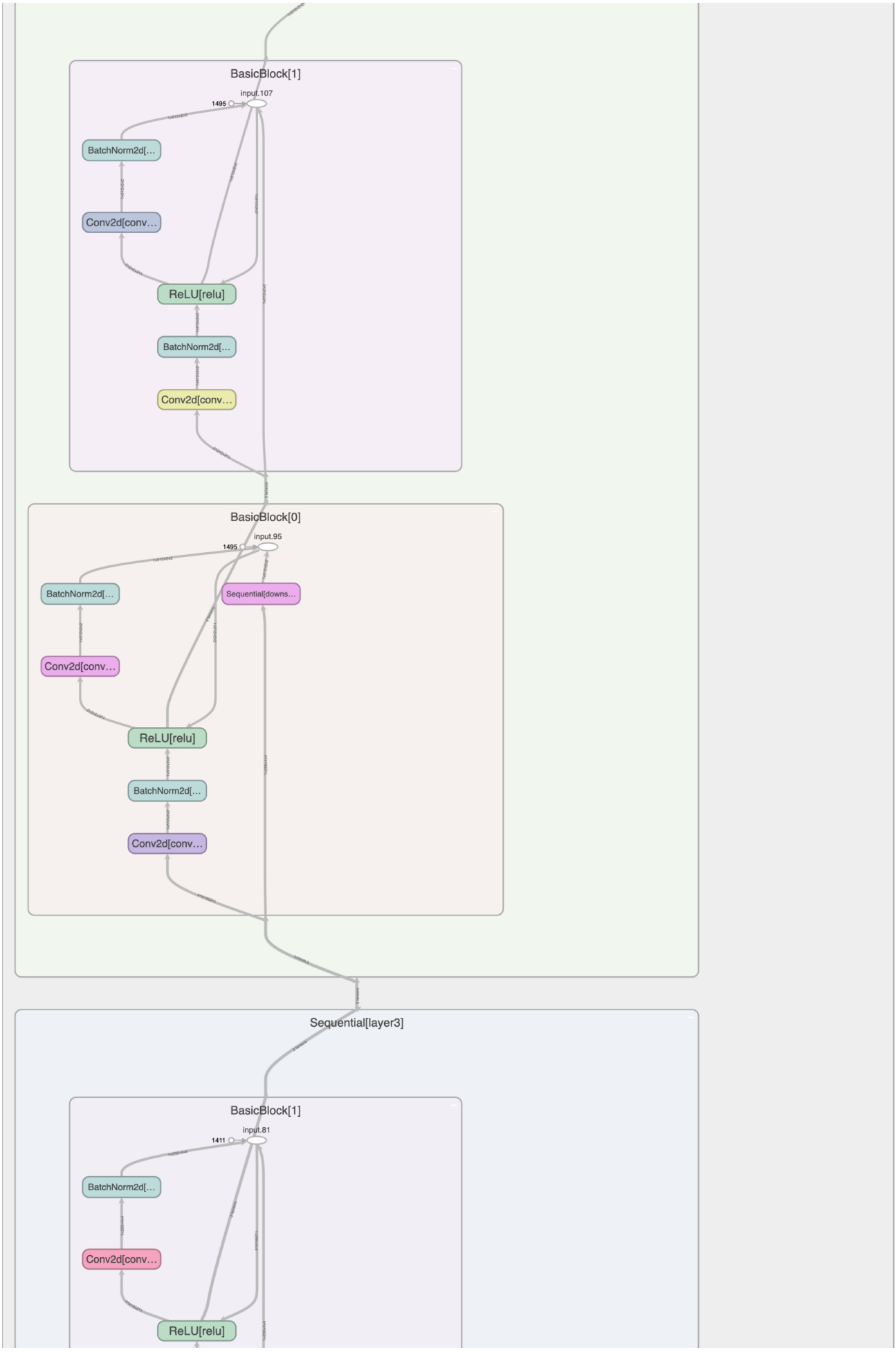
一、计算 Resnet18 各层处理大小

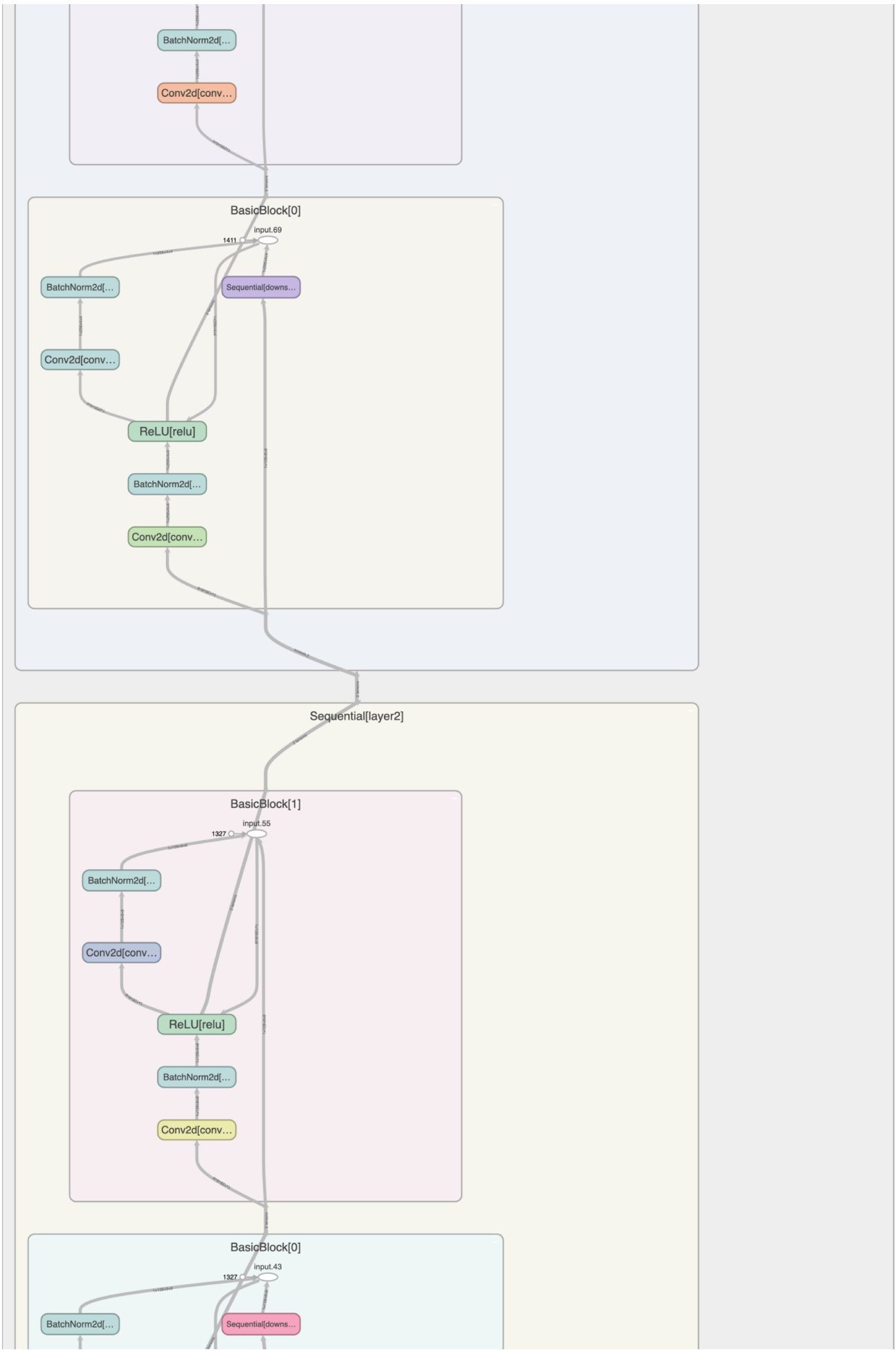
- input：3 * 64 * 64
- conv1：64 * 32 * 32
- bn1：64 * 32 * 32
- relu：64 * 32 * 32
- maxpool：64 * 16 * 16
- layer1
  - BasicBlock0
    - conv1：64 * 16 * 16
    - bn1：64 * 16 * 16
    - relu：64 * 16 * 16
    - conv2：64 * 16 * 16
    - bn2：64 * 16 * 16
  - BasicBlock1
    - conv1：64 * 16 * 16
    - bn1：64 * 16 * 16
    - relu：64 * 16 * 16
    - conv2：64 * 16 * 16
    - bn2：64 * 16 * 16
- layer2
  - BasicBlock0
    - conv1：128 * 8 * 8

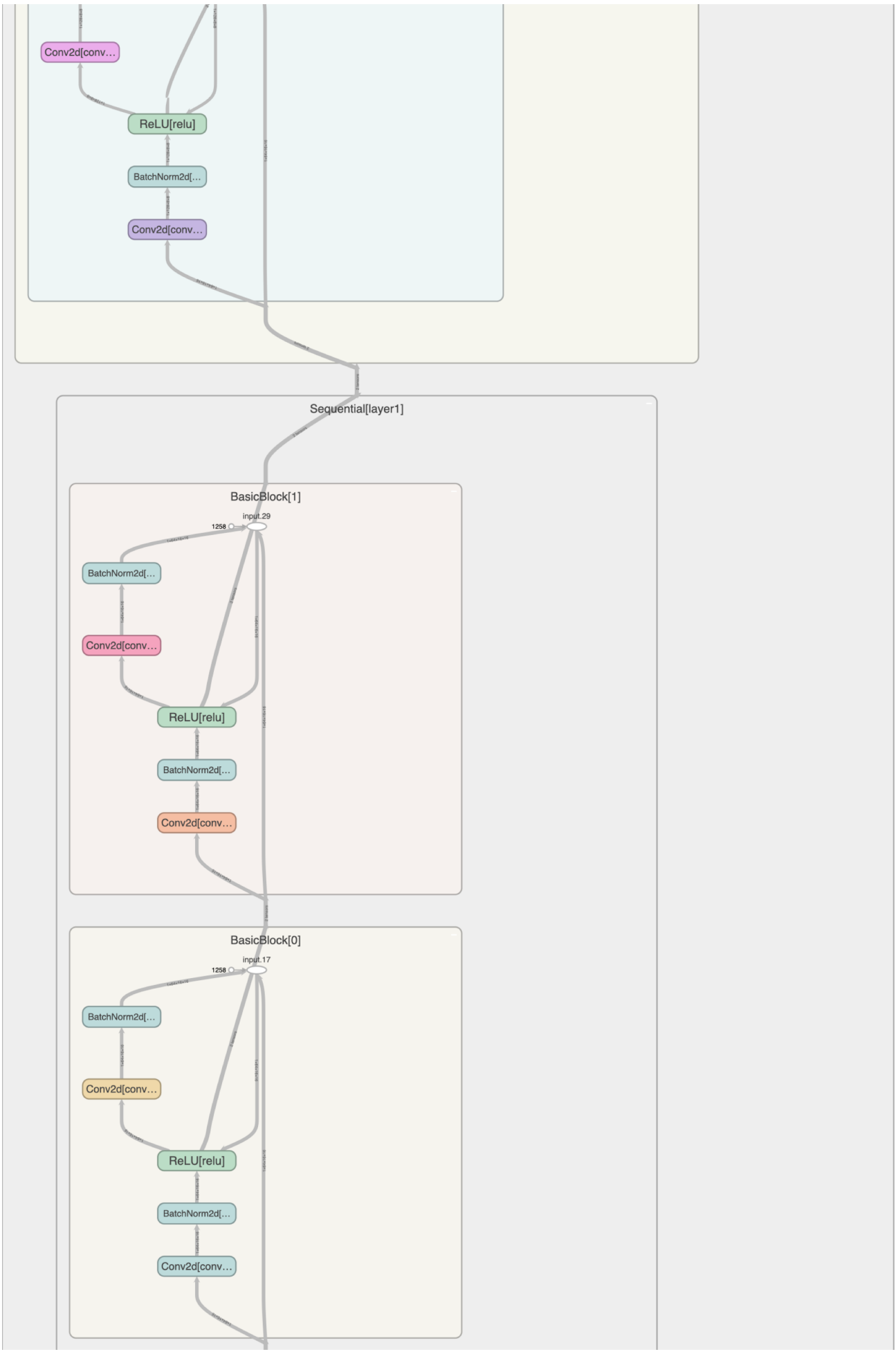
        - bn1：128 * 8 * 8
        - relu：128 * 8 * 8
        - conv2：128 * 8 * 8
        - bn2：128 * 8 * 8
        - donwsample：
    - BasicBlock1
        - conv1：128 * 8 * 8
        - bn1：128 * 8 * 8
        - relu：128 * 8 * 8
        - conv2：128 * 8 * 8
        - bn2：128 * 8 * 8
- layer3
    - BasicBlock0
        - conv1：256 * 4 * 4
        - bn1：256 * 4 * 4
        - relu：256 * 4 * 4
        - conv2：256 * 4 * 4
        - bn2：256 * 4 * 4
        - downsample：256 * 4 * 4
    - BasicBlock1
        - conv1：256 * 4 * 4
        - bn1：256 * 4 * 4
        - relu：256 * 4 * 4
        - conv2：256 * 4 * 4

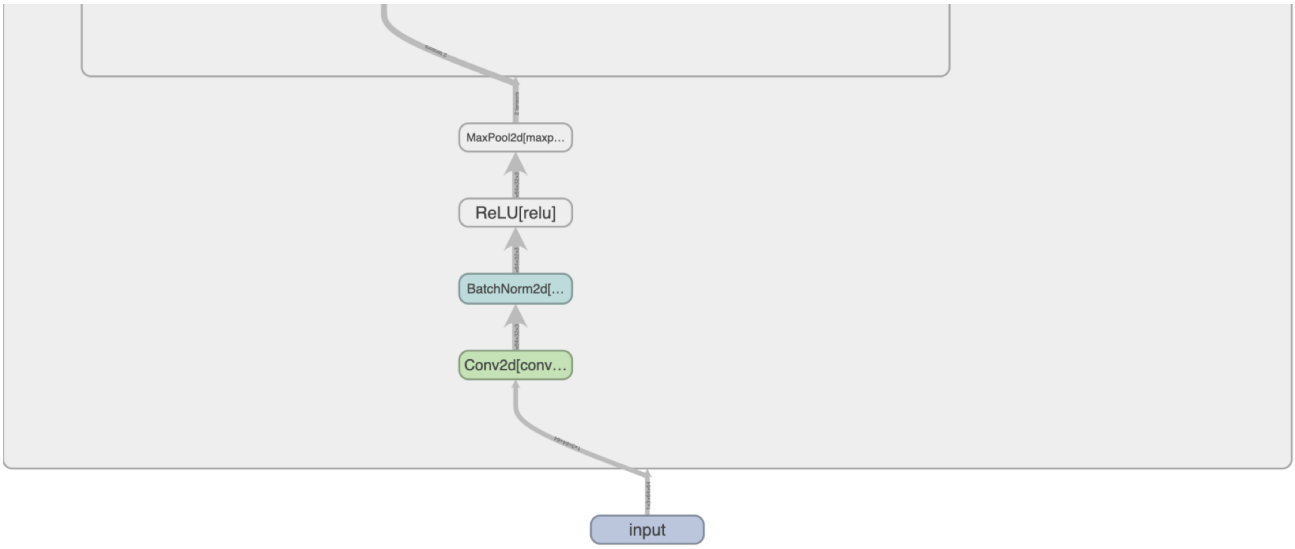
        - bn2：256 * 4 * 4
- layer4
    - BasicBlock0
        - conv1：512 * 2 * 2
        - bn1：512 * 2 * 2
        - relu：512 * 2 * 2
        - conv2：512 * 2 * 2
        - bn2：512 * 2 * 2
        - downsample：512 * 2 * 2
    - BasicBlock1
        - conv1：512 * 2 * 2
        - bn1：512 * 2 * 2
        - relu：512 * 2 * 2
        - conv2：512 * 2 * 2
        - bn2：512 * 2 * 2
- avgpool：512 * 1 * 1
- fc：1000
- output：1000




BasicBlock[1]
input.107
1495
BatchNorm2d[…
Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…
BasicBlock[0]
input.95
1495
BatchNorm2d[…
Sequential[downs…
Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…
Sequential[layer3]
BasicBlock[1]
input.81
1411
BatchNorm2d[…
Conv2d[conv…
ReLU[relu]


BatchNorm2d[…
Conv2d[conv…
BasicBlock[0]
input.69
1411
BatchNorm2d[…
Sequential[downs…
Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…
Sequential[layer2]
BasicBlock[1]
input.55
1327
BatchNorm2d[…
Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…
BasicBlock[0]
input.43
1327
BatchNorm2d[…
Sequential[downs…


Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…
Sequential[layer1]
BasicBlock[1]
input.29
1258
BatchNorm2d[…
Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…
BasicBlock[0]
input.17
1258
BatchNorm2d[…
Conv2d[conv…
ReLU[relu]
BatchNorm2d[…
Conv2d[conv…



二、代码 `main.py` 的改写

- 导入 `SummaryWriter` 类

```
@@ -19,6 +19,11 @@ import torch.utils.data.distributed
 import torchvision.transforms as transforms
 import torchvision.datasets as datasets
 import torchvision.models as models
+from tensorboardX import SummaryWriter
+
+
+# 将 loss 和 accuracy 保存到目录 ./runs 中
+writer = SummaryWriter()
```

- 修改网络输出类别，并修改对图片进行伸缩和裁剪的代码

```
            model = models.__dict__[args.arch]()

+    # 修改网络结构，输出类别为200
+    in_features = model.fc.in_features
+    model.fc = nn.Linear(in_features, 200)
+
     if not torch.cuda.is_available():
         print('using CPU, this will be slow')
     elif args.distributed:
@@ -213,7 +222,7 @@ def main_worker(gpu, ngpus_per_node, args):
     train_dataset = datasets.ImageFolder(
         traindir,
         transforms.Compose([
-            transforms.RandomResizedCrop(224),
+            # transforms.RandomResizedCrop(224),
             transforms.RandomHorizontalFlip(),
             transforms.ToTensor(),
             normalize,
@@ -230,8 +239,8 @@ def main_worker(gpu, ngpus_per_node, args):

     val_loader = torch.utils.data.DataLoader(
         datasets.ImageFolder(valdir, transforms.Compose([
-            transforms.Resize(256),
-            transforms.CenterCrop(224),
+            # transforms.Resize(256),
+            # transforms.CenterCrop(224),
             transforms.ToTensor(),
             normalize,
         ])),
@@ -247,13 +256,20 @@ def main_worker(gpu, ngpus_per_node, args):
:
```

- 记录训练过程中的训练集和验证集的损失和精度变化

```
        # train for one epoch
-       train(train_loader, model, criterion, optimizer, epoch, args)
+       loss, acc5 = train(train_loader, model, criterion, optimizer, epoch, args)
+
+       # record train loss and accuracy
+       writer.add_scalar('Loss/train', loss, epoch)
+       writer.add_scalar('Accuracy/train', acc5, epoch)

        # evaluate on validation set
-       acc1 = validate(val_loader, model, criterion, args)
+       loss, acc1, acc5 = validate(val_loader, model, criterion, args)
+
+       # record validation loss and accuracy
+       writer.add_scalar('Loss/val', loss, epoch)
+       writer.add_scalar('Accuracy/val', acc5, epoch)

        scheduler.step()
-

        # remember best acc@1 and save checkpoint
        is_best = acc1 > best_acc1
@@ -270,6 +286,8 @@ def main_worker(gpu, ngpus_per_node, args):
                'scheduler' : scheduler.state_dict()
            }, is_best)

+    writer.close()
+
```

- 修改对 `函数train` 和 `validata` 的定义，返回损失和精度

```
 def train(train_loader, model, criterion, optimizer, epoch, args):
     batch_time = AverageMeter('Time', ':6.3f')
args):
         if i % args.print_freq == 0:
             progress.display(i)

+    return losses.avg, top5.avg
+

 def validate(val_loader, model, criterion, args):
     batch_time = AverageMeter('Time', ':6.3f', Summary.NONE)
@@ -358,7 +378,7 @@ def validate(val_loader, model, criterion, args):

         progress.display_summary()

-    return top1.avg
+    return losses.avg, top1.avg, top5.avg
```

三、验证数据集的代码编写

编写脚本将验证集的数据目录结构更改为与训练集一致，结果保存在`val.py`中，实现思路如下：

- 定义一个集合变量`val_dict`用来保存每张图片的以`.JPEG`结尾的名称和该图片的整数值标签
- 遍历所有图片，将图片名称和标签保存到`val_dict`中
- 再次遍历所有图片，以每张图片的标签为目录名创建文件夹，如果该文件夹存在就跳过
- 最后遍历一次所有图片，将每张图片按标签移动到对应文件夹中
- 删除原始文件夹

关键步骤实现如下：

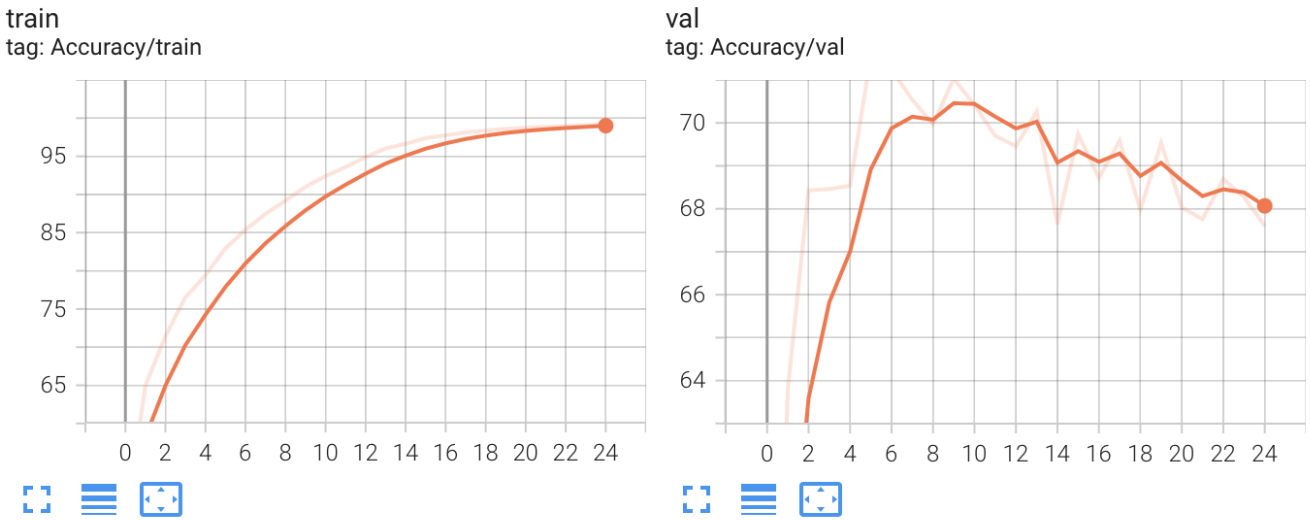
```
target_folder = './tiny-imagenet-200/val/'

val_dict = {}

with open('./tiny-imagenet-200/val/val_annotations.txt', mode='r') as f:

    for line in f.readlines():

        line = line.split('\t')

        val_dict[line[0]] = line[1]

paths = glob.glob('./tiny-imagenet-200/val/images/*')

for path in paths:

    file = path.split('/')[-1]

    folder = val_dict[file]

    if not os.exsists(target_folder + str(folder)):

        os.mkdir(target_folder + str(folder))

        os.mkdir(target_folder + str(folder) + '/images')

for path in paths:

    file = path.split('/')[-1]

    folder = val_dict[file]

    dest = target_folder + str(folder) + '/images/' + str(file)

    shutil.move(path, dest)

os.rmdir('./tiny-imagesnet-200/val/images')
```

四、训练

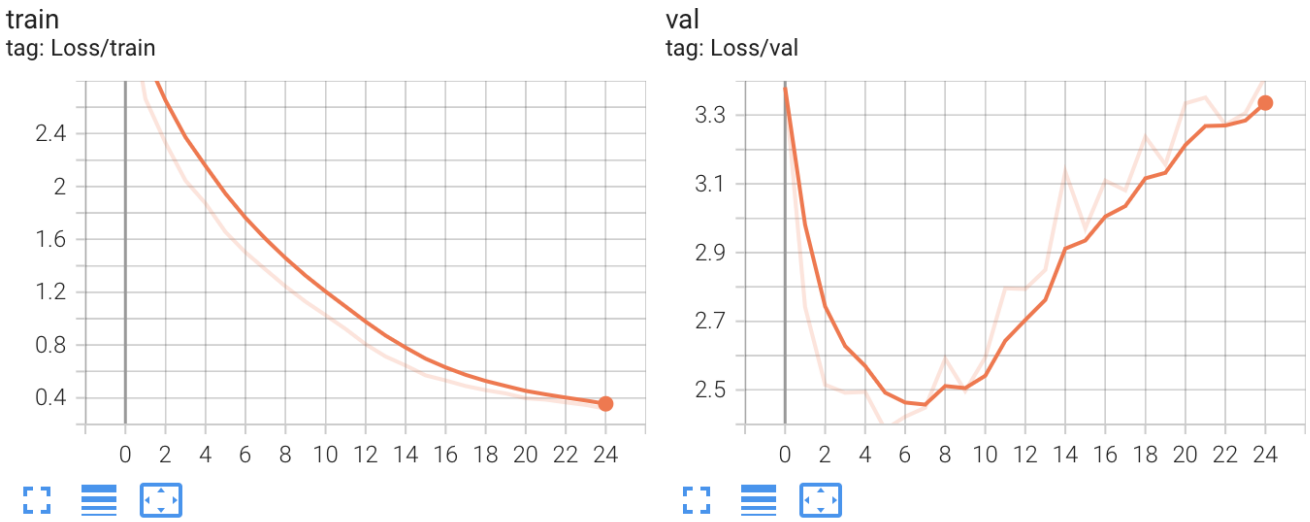
启动命令`python main.py -a resnet18 --epochs 25 --pretrained ./tiny-imagenet-200`，学习率 lr 、batch_size 、动量 momentum 等超参数保持默认。

训练结果曲线如下：

Accuracy



Loss



由截图可以看出，随着 epoch 的迭代，训练集的损失在下降，而训练精度在上升。然而对于验证集而言，损失曲线和精度曲线都呈现了转折，即在 epoch = 7 之前，验证集损失在下降，且精度在上升，并在 epoch=7 达到极值点，这个趋势与训练集相同，但当 epoch 超过 7 之后，验证集的损失反而在上升，且精度在下降。因此，随着 epoch 的继续增加，网络在训练集上表现良好，但在验证集上的表现不佳，出现了过拟合现象。

五、模型 checkpoint 的保存和比较

保存模型的两个 checkpoint `checkpoint.pht.tar` 和 `model_best.pth.tar`，运行命令 `python main.py --evaluate --resume ./checkpoint.pth.tar ./tiny-imagenet-200` 和 `python main.py --evaluate --resume ./model_best.pth.tar ./tiny-imagenet-200`，得到如下模型比较结果：

```
PS D:\ustc\dl> conda activate base
PS D:\ustc\dl> python main.py --evaluate --resume .\checkpoint.pth.tar .\tiny-imagenet-200\
=> creating model 'resnet18'
=> loading checkpoint '.\checkpoint.pth.tar'
=> loaded checkpoint '.\checkpoint.pth.tar' (epoch 25)
Test: [ 0/40]   Time 11.926 (11.926)    Loss 2.9837e+00 (2.9837e+00)    Acc@1  46.48 ( 46.48)   Acc@5  69.92 ( 69.92)
Test: [10/40]   Time  0.154 ( 1.295)    Loss 3.0164e+00 (3.2415e+00)    Acc@1  46.88 ( 43.82)   Acc@5  71.48 ( 69.21)
Test: [20/40]   Time  0.069 ( 0.895)    Loss 3.2777e+00 (3.4222e+00)    Acc@1  41.80 ( 41.20)   Acc@5  66.02 ( 67.08)
Test: [30/40]   Time  0.234 ( 0.814)    Loss 3.8002e+00 (3.4526e+00)    Acc@1  35.94 ( 41.23)   Acc@5  68.36 ( 66.89)
 *   Acc@1 41.590 Acc@5 67.600
PS D:\ustc\dl> python main.py --evaluate --resume .\model_best.pth.tar .\tiny-imagenet-200\
=> creating model 'resnet18'
=> loading checkpoint '.\model_best.pth.tar'
=> loaded checkpoint '.\model_best.pth.tar' (epoch 10)
Test: [ 0/40]   Time  9.331 ( 9.331)    Loss 2.1208e+00 (2.1208e+00)    Acc@1  48.44 ( 48.44)   Acc@5  73.83 ( 73.83)
Test: [10/40]   Time  0.068 ( 0.913)    Loss 2.4185e+00 (2.3257e+00)    Acc@1  48.83 ( 47.12)   Acc@5  73.83 ( 73.72)
Test: [20/40]   Time  0.068 ( 0.512)    Loss 2.5048e+00 (2.5350e+00)    Acc@1  46.88 ( 44.83)   Acc@5  69.92 ( 70.41)
Test: [30/40]   Time  0.070 ( 0.370)    Loss 2.6164e+00 (2.5257e+00)    Acc@1  45.31 ( 45.09)   Acc@5  71.09 ( 70.48)
 *   Acc@1 45.090 Acc@5 71.030
PS D:\ustc\dl>
```

可以看出，对于`checkpoint.pth.tar`，虽然训练的 epochs 更多，在训练集上的精度更高，但由于出现了过拟合现象，因此在测试集上的精度不高，平均只有68%左右。而对于`model_bets.pth.tar`，虽然只训练了10个 epochs，但在测试集上的精度接近最优，平均达到了72%左右。