

Vision Modeling Lab

과제수행 4회차

ICT 융합학부

2019098068

이찬영

MultiMNIST 1

```
1 import torch
2 from torchvision import transforms
3 from MultiMnist_v1_2 import MultiMNIST
4
5 import torch.nn.functional as F
6 import torch.nn as nn
7 import torch.nn.init as init
8 import matplotlib.pyplot as plt
9 import numpy as np
10 import wandb
11
12 wandb.init(project='multimnist')
13 wandb.run.name = 'resnet_01'
14 wandb.run.save()
15
16 # 딥러닝 모델을 설계할 때 활용하는 GPU
17 if torch.cuda.is_available():
18     DEVICE = torch.device('cuda')
19 else:
20     DEVICE = torch.device('cpu')
21 print(DEVICE)
22
23 batch_size = 128      # 데이터 개수
24 EPOCHS = 300          # 학습 횟수
25
26 image_transform = transforms.Compose([
27     transforms.RandomHorizontalFlip(),
28     transforms.ToTensor(),      # 이미지를 텐서로 변환
29     transforms.Normalize((0.5), (0.5))])
30
31 # MNIST 데이터 갖고 오기
32 train_dataset = MultiMNIST(train=True, rot_mode=True, size_mode=True, noise_mode=True, transform=image_transform)
```

MultiMNIST 1

```
33 test_dataset = MultiMNIST(train=False, rot_mode=True, size_mode=True, noise_mode=True, transform=image_transform)
34 # 다운로드한 MNIST 데이터셋을 batch_size 단위로 분리해 저장
35 train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
37
38 # Residual block
39 class BasicBlock(nn.Module):
40     def __init__(self, in_planes, planes, stride=1):
41         super(BasicBlock, self).__init__()
42         self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
43         self.bn1 = nn.BatchNorm2d(planes)
44         self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
45         self.bn2 = nn.BatchNorm2d(planes)
46
47         self.shortcut = nn.Sequential() # Resnet Shortcut (기존의 값과 CNN+BN 한 결과를 더함)
48         if stride != 1 or in_planes != planes:
49             self.shortcut = nn.Sequential(
50                 nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=False),
51                 nn.BatchNorm2d(planes))
52
53     def forward(self, x): # forward propagation
54         out = F.relu(self.bn1(self.conv1(x))) # conv -> bn1 -> relu -> conv2 -> bn2
55         out = self.bn2(self.conv2(out))
56         out += self.shortcut(x) # skip connection
57         out = F.relu(out)
58         return out
59
60 # Resnet 모델
61 class ResNet(nn.Module):
62     def __init__(self, num_classes=100):
63         super(ResNet, self).__init__()
64         self.in_planes = 16
```

MultiMNIST 1

ResNet18 적용

```

65
66 # input block
67 self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1, bias=False)
68 self.bn1 = nn.BatchNorm2d(16)
69
70 # residual blocks
71 self.layer1 = self._make_layer(planes=16, num_blocks=2, stride=1)
72 self.layer2 = self._make_layer(planes=64, num_blocks=2, stride=2)
73 self.layer3 = self._make_layer(planes=128, num_blocks=2, stride=2)
74 self.layer4 = self._make_layer(planes=256, num_blocks=2, stride=2)
75 self.fc = nn.Linear(in_features=256, num_classes)
76
77 def _make_layer(self, planes, num_blocks, stride):
78     strides = [stride] + [1] * (num_blocks - 1)
79     layers = []
80     for stride in strides:
81         layers.append(BasicBlock(self.in_planes, planes, stride))
82     self.in_planes = planes
83     return nn.Sequential(*layers)
84
85 def forward(self, x):
86     out = F.relu(self.bn1(self.conv1(x)))
87     out = self.layer1(out)
88     out = self.layer2(out)
89     out = self.layer3(out)
90     out = self.layer4(out)
91     out = F.avg_pool2d(out, 8)
92     out = out.view(out.size(0), -1)
93     out = self.fc(out) # Fully Connected Layer
94     return out
95
96 def weight_init(m):

```

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10 ⁹	3.6×10 ⁹	3.8×10 ⁹	7.6×10 ⁹	11.3×10 ⁹

tures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

MultiMNIST 1

```
97     if isinstance(m, nn.Linear):
98         init.kaiming_uniform_(m.weight.data)
99
100 # Initialize
101 model = ResNet().to(DEVICE)
102 model.apply(weight_init)
103 optimizer = torch.optim.Adam(model.parameters(), lr=.0001) # 학습률 : 모델의 가중치를 0.0001만큼 조정
104 criterion = nn.CrossEntropyLoss() # 다중 분류를 위한 손실 함수
105 print(model)
106
107 # 이미지 데이터와 레이블 데이터를 이용해 MLP 모델을 학습
108 def train(model, train_loader, optimizer):
109     model.train() # MLP 모델을 학습 상태로 지정
110     total_loss = 0.
111     n = 0
112
113     for batch_idx, (image, label, bb) in enumerate(train_loader): # 기존에 정의한 GPU에 데이터를 할당
114         image = image.to(DEVICE)
115         label = label.to(DEVICE)
116
117         label_temp = torch.zeros(label.shape[0], 1).to(DEVICE)
118         label_temp = label[:, 0] * 10 + label[:, 1]
119
120         optimizer.zero_grad() # optimizer Gradient 초기화
121         # Forward
122         output = model(image) # Input : 이미지 데이터 / Output 계산
123         # Backward
124         loss = criterion(output, label_temp) # loss 값 계산
125         loss.backward() # Back 통해 계산된 Gradient 값을 각 파라미터에 할당함
126         optimizer.step() # 각 파라미터에 할당된 Gradient 값을 이용해 파라미터 값을 업데이트함
127         total_loss += loss.item() # Propagation을 각 batch에 대한 손실값을 더해줌 (scalar)
128
```

MultiMNIST 1

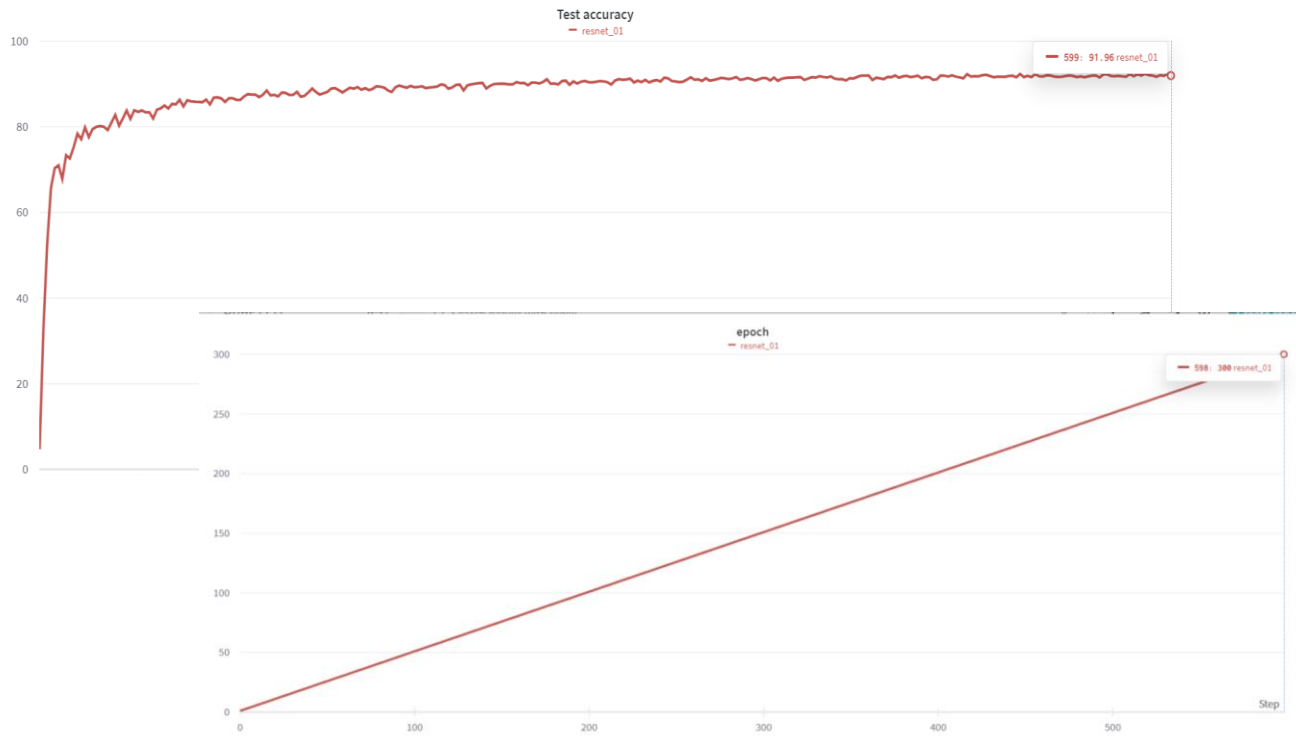
```
129         n += (output1.shape[0] + output2.shape[0])    # 각 batch에 포함된 샘플 수를 더해줌
130
131     print(f"Train Epoch: {epoch}, Train Loss: {total_loss / n:.6f}")
132     args = {
133         "Train Loss": total_loss / n,
134         "epoch": epoch
135     }
136     wandb.log(args)
137
138 # 검증 데이터에 대한 모델 성능을 확인
139 def evaluate(model, test_loader):
140     model.eval()    # MLP 모델을 평가 상태로 지정
141     test_loss = 0
142     correct = 0
143     n = 0
144
145     with torch.no_grad():    # 파라미터 업데이트 방지
146         for image, label, bb in test_loader:    # image, label, bounding_box
147             image = image.to(DEVICE)
148             label = label.to(DEVICE)
149
150             label_temp = torch.zeros(label.shape[0], 1).to(DEVICE)
151             label_temp = label[:, 0] * 10 + label[:, 1]
152
153             output = model(image)
154             test_loss += criterion(output, label_temp).item()    # loss 값 계산
155             prediction = output.max(1, keepdim=True)[1]    # 계산된 벡터값 내 가장 큰 값인 위치에 대해 해당 위치에 대응하는 클래스로 예측
156             correct += prediction.eq(label_temp.view_as(prediction)).sum().item()    # 예측한 클래스 값과 실제 레이블이 의미하는 클래스가 맞으면 correct에 더해 올바르게 예측한 횟수를 저장
157
158     test_loss /= len(test_loader.dataset)    # 평균 loss 값
159     test_accuracy = 100. * correct / len(test_loader.dataset)    # test_loader 데이터 중 얼마나 맞췄는지를 계산해 정확도를 계산함
160     wandb.log({
161         "Test loss": test_loss,
```

MultiMNIST 1

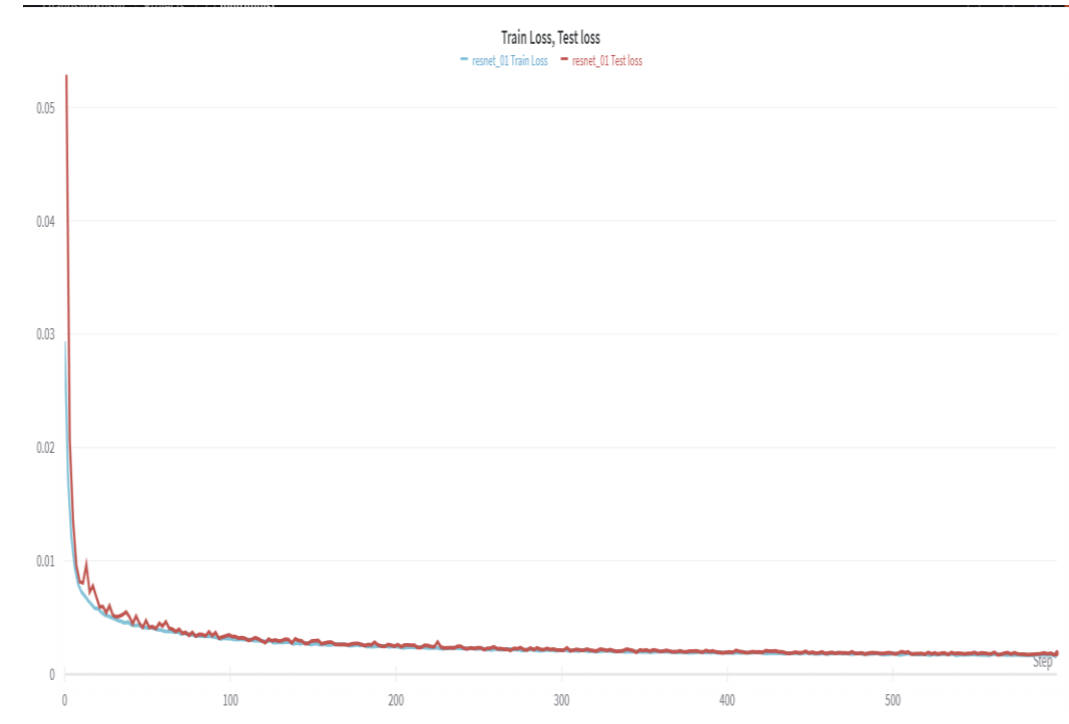
```
161         "Test accuracy": test_accuracy
162     })
163     return test_loss, test_accuracy
164
165     for epoch in range(1, EPOCHS + 1):
166         train(model, train_loader, optimizer)
167         ⚡ test_loss, test_accuracy = evaluate(model, test_loader)
168         print(f"\n[EPOCH: {epoch}], \tTest Loss: {test_loss:.4f}, \tTest Accuracy: {test_accuracy:.2f} %\n")
```

```
[EPOCH: 297],   Test Loss: 0.0018, Test Accuracy: 92.08 %
Train Epoch: 298, Train Loss: 0.001699
[EPOCH: 298],   Test Loss: 0.0018, Test Accuracy: 91.90 %
Train Epoch: 299, Train Loss: 0.001713
[EPOCH: 299],   Test Loss: 0.0017, Test Accuracy: 92.46 %
Train Epoch: 300, Train Loss: 0.001696
[EPOCH: 300],   Test Loss: 0.0019, Test Accuracy: 91.96 %
```

300 epoch



Resnet18 적용 시 -> Test Accuracy : **92.46 %**



MultimNIST 1

```
(l1): BasicBlock{
  (conv1): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (shortcut): Sequential{
  }
}

(layer3): Sequential{
  (l0): BasicBlock{
    (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (shortcut): Sequential{
      (l0): Conv2d(32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (l1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    }
  }
  (l1): BasicBlock{
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (shortcut): Sequential{
    }
  }
}

(layer4): Sequential{
  (l0): BasicBlock{
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (shortcut): Sequential{
    }
  }
}
```

```

    )
    (l1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer5): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
  (l1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (shortcut): Sequential()
  )
)
(fc): Linear(in_features=256, out_features=100, bias=True)
)

```

Resnet14

Test Accuracy : 88.24 % | Test Accuracy : 92.46 % (300 epoch)

```

145 test_loss /= len(test_loader.dataset) # 평균 loss 값
146 test_accuracy = 100. * correct / len(test_loader.dataset)
147 tl2.append(round(test_loss, 6))
148 al1.append(test_accuracy)
149 return test_loss, test_accuracy
150
151 tl1 = [] # Train loss
152 tl2 = [] # Test loss
153 al1 = [] # Accuracy
154 for epoch in range(1, EPOCHS + 1):

```

Run main × main (2) ×

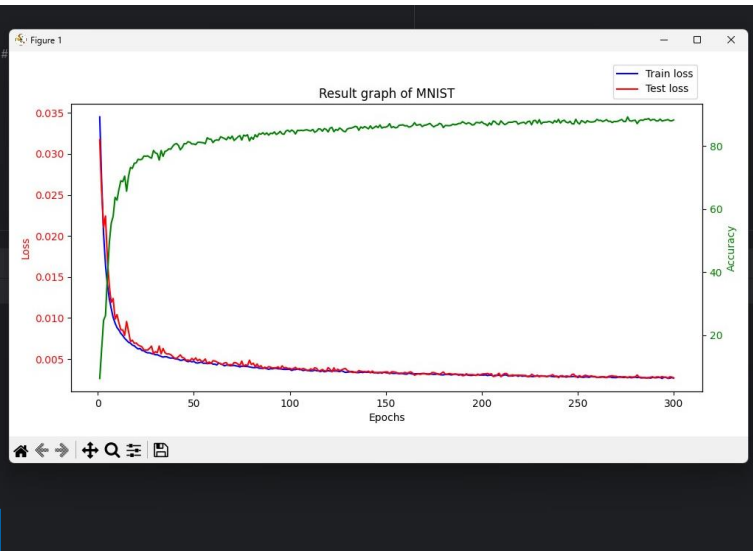
```

Train Epoch: 298, Train Loss: 0.002670
[EPPOCH: 298], Test Loss: 0.0028, Test Accuracy: 88.02 %

Train Epoch: 299, Train Loss: 0.002745
[EPPOCH: 299], Test Loss: 0.0028, Test Accuracy: 87.98 %

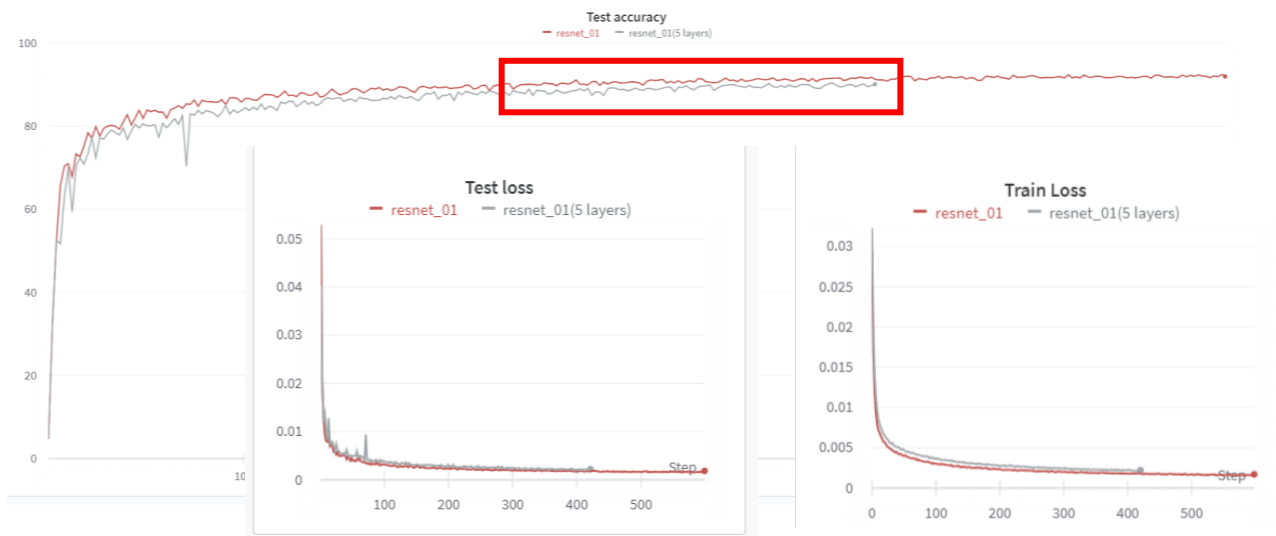
Train Epoch: 300, Train Loss: 0.002672
[EPPOCH: 300], Test Loss: 0.0027, Test Accuracy: 88.24 %

```

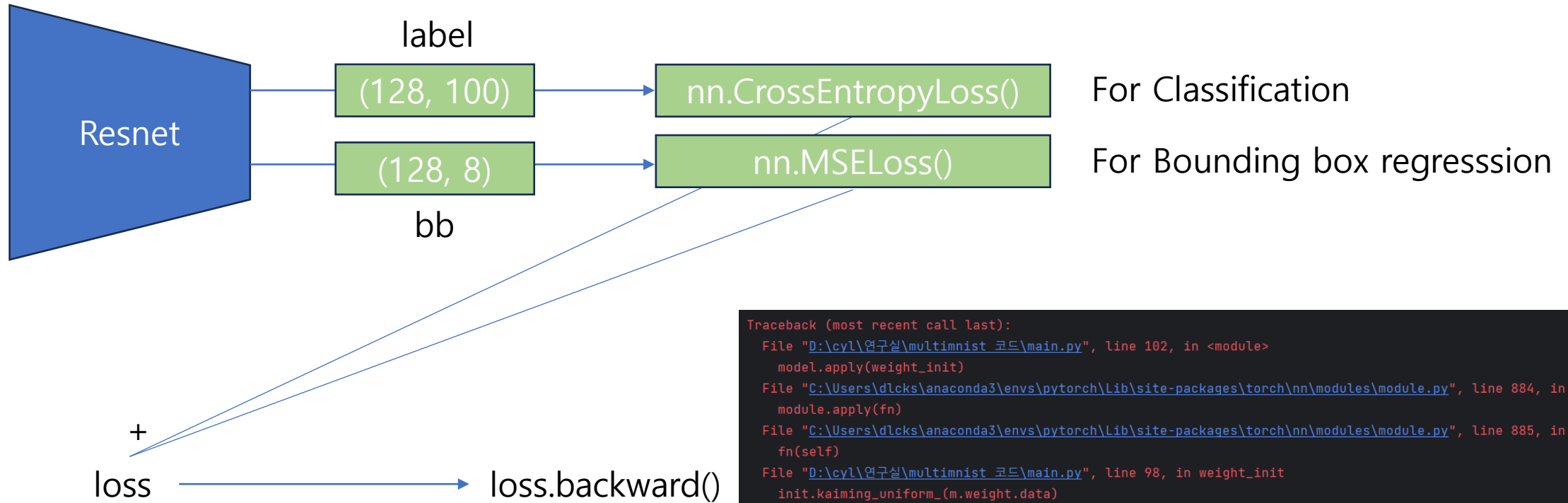


Resnet22

Test Accuracy : 90.44 % | Test Accuracy : 91.5 % (200 epoch)



MultiMNIST 1



```
Traceback (most recent call last):
  File "D:\cyl\연구실\multimnist 코드\main.py", line 102, in <module>
    model.apply(weight_init)
  File "C:\Users\dicks\anaconda3\envs\pytorch\Lib\site-packages\torch\nn\modules\module.py", line 884, in apply
    module.apply(fn)
  File "C:\Users\dicks\anaconda3\envs\pytorch\Lib\site-packages\torch\nn\modules\module.py", line 885, in apply
    fn(self)
  File "D:\cyl\연구실\multimnist 코드\main.py", line 98, in weight_init
    init.kaiming_uniform_(m.weight.data)
  File "C:\Users\dicks\anaconda3\envs\pytorch\Lib\site-packages\torch\nn\init.py", line 412, in kaiming_uniform_
    return tensor.uniform_(bound, bound)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
RuntimeError: CUDA error: out of memory

CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect.
For debugging consider passing CUDA_LAUNCH_BLOCKING=1.

Compile with `TORCH_USE_CUDA_DSA` to enable device-side assertions.
```