Vision Modeling Lab

과제수행 4회차

ICT 융합학부
2019098068
이찬영

MultiMNIST 1

```python
import torch
from torchvision import transforms
from MultiMnist_v_1_2 import MultiMNIST

import torch.nn.functional as F
import torch.nn as nn
import torch.nn.init as init
import matplotlib.pyplot as plt
import numpy as np
import wandb

wandb.init(project='multimnist')
wandb.run.name = 'resnet_01'
wandb.run.save()

# 딥러닝 모델을 설계할 때 활용하는 GPU
if torch.cuda.is_available():
    DEVICE = torch.device('cuda')
else:
    DEVICE = torch.device('cpu')
print(DEVICE)

batch_size = 128        # 데이터 개수
EPOCHS = 300            # 학습 횟수

image_transform = transforms.Compose([
                    transforms.RandomHorizontalFlip(),
                    transforms.ToTensor(),     # 이미지를 텐서로 변환
                    transforms.Normalize((0.5), (0.5))])

# MNIST 데이터 갖고 오기
train_dataset = MultiMNIST(train=True, rot_mode=True, size_mode=True, noise_mode=True, transform=image_transform)
```

MultiMNIST 1

```python
test_dataset = MultiMNIST(train=False, rot_mode=True, size_mode=True, noise_mode=True, transform=image_transform)
# 다운로드한 MNIST 데이터셋을 batch_size 단위로 분리해 저장
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)

# Residual block
class BasicBlock(nn.Module):
    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()         # Resnet Shortcut (기존의 값과 CNN+BN 한 결과를 더함)
        if stride != 1 or in_planes != planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(planes))

    def forward(self, x):       # forward propagation
        out = F.relu(self.bn1(self.conv1(x)))   # conv -> bn1 -> relu -> conv2 -> bn2
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)     # skip connection
        out = F.relu(out)
        return out

# Resnet 모델
class ResNet(nn.Module):
    def __init__(self, num_classes=100):
        super(ResNet, self).__init__()
        self.in_planes = 16
```

MultiMNIST 1

ResNet18 적용

```python

        # input block
        self.conv1 = nn.Conv2d( in_channels: 1,  out_channels: 16, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(16)

        # residual blocks
        self.layer1 = self._make_layer( planes: 16,  num_blocks: 2, stride=1)
        self.layer2 = self._make_layer( planes: 64,  num_blocks: 2, stride=2)
        self.layer3 = self._make_layer( planes: 128,  num_blocks: 2, stride=2)
        self.layer4 = self._make_layer( planes: 256,  num_blocks: 2, stride=2)
        self.fc = nn.Linear( in_features: 256, num_classes)

    def _make_layer(self, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(BasicBlock(self.in_planes, planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 8)
        out = out.view(out.size(0), -1)
        out = self.fc(out)  # Fully Connected Layer
        return out

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	[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×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

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

MultiMNIST 1

```python
    if isinstance(m, nn.Linear):
        init.kaiming_uniform_(m.weight.data)

# Initialize
model = ResNet().to(DEVICE)
model.apply(weight_init)
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)   # 학습률 : 모델의 가중치를 0.0001만큼 조정
criterion = nn.CrossEntropyLoss()    # 다중 분류를 위한 손실 함수
print(model)

# 이미지 데이터와 레이블 데이터를 이용해 MLP 모델을 학습
def train(model, train_loader, optimizer):
    model.train()       # MLP 모델을 학습 상태로 지정
    total_loss = 0.
    n = 0

    for batch_idx, (image, label, bb) in enumerate(train_loader): # 기존에 정의한 GPU에 데이터를 할당
        image = image.to(DEVICE)
        label = label.to(DEVICE)

        label_temp = torch.zeros(label.shape[0], 1).to(DEVICE)
        label_temp = label[:, 0] * 10 + label[:, 1]

        optimizer.zero_grad()       # optimizer Gradient 초기화
        # Forward
        output = model(image)       # Input : 이미지 데이터 / Output 계산
        # Backward
        loss = criterion(output1, label_temp)     # loss 값 계산
        loss.backward()     # Back 통해 계산된 Gradient 값을 각 파라미터에 할당함
        optimizer.step()    # 각 파라미터에 할당된 Gradient 값을 이용해 파라미터 값을 업데이트함
        total_loss += loss.item()   # Propagation을 각 batch에 대한 손실값을 더해줌 (scalar)

```

MultiMNIST 1

```
            n += (output1.shape[0] + output2.shape[0])    # 각 batch에 포함된 샘플 수를 더해줌

    print(f"Train Epoch: {epoch}, Train Loss: {total_loss / n:.6f}")
    args = {
        "Train Loss": total_loss / n,
        "epoch": epoch
    }
    wandb.log(args)

# 검증 데이터에 대한 모델 성능을 확인
def evaluate(model, test_loader):
    model.eval()    # MLP 모델을 평가 상태로 지정
    test_loss = 0
    correct = 0
    n = 0

    with torch.no_grad():       # 파라미터 업데이트 방지
        for image, label, bb in test_loader:      # image, label, bounding_box
            image = image.to(DEVICE)
            label = label.to(DEVICE)

            label_temp = torch.zeros(label.shape[0], 1).to(DEVICE)
            label_temp = label[:, 0] * 10 + label[:, 1]

            output = model(image)
            test_loss += criterion(output, label_temp).item()    # loss 값 계산
            prediction = output.max(1, keepdim = True)[1]    # 계산된 벡터값 내 가장 큰 값인 위치에 대해 해당 위치에 대응하는 클래스로 예측
            correct += prediction.eq(label_temp.view_as(prediction)).sum().item()   # 예측한 클래스 값과 실제 레이블이 의미하는 클래스가 맞으면 correct에 더해 올바르게 예측한 횟수를 저장
    test_loss /= len(test_loader.dataset)        # 평균 loss 값
    test_accuracy = 100. * correct / len(test_loader.dataset)   # rest_loader 데이터 중 얼마나 맞췄는지를 계산해 정확도를 계산함
    wandb.log({
        "Test loss" : test_loss,
```

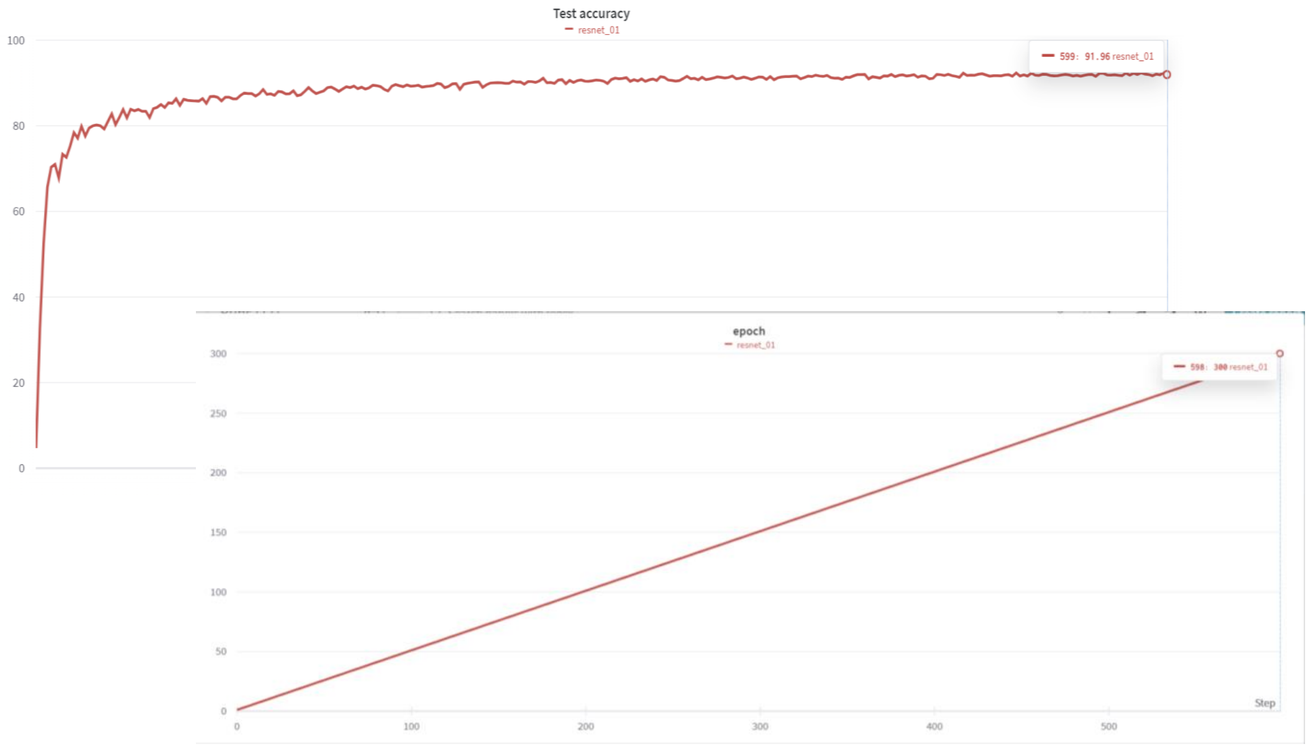
MultiMNIST 1

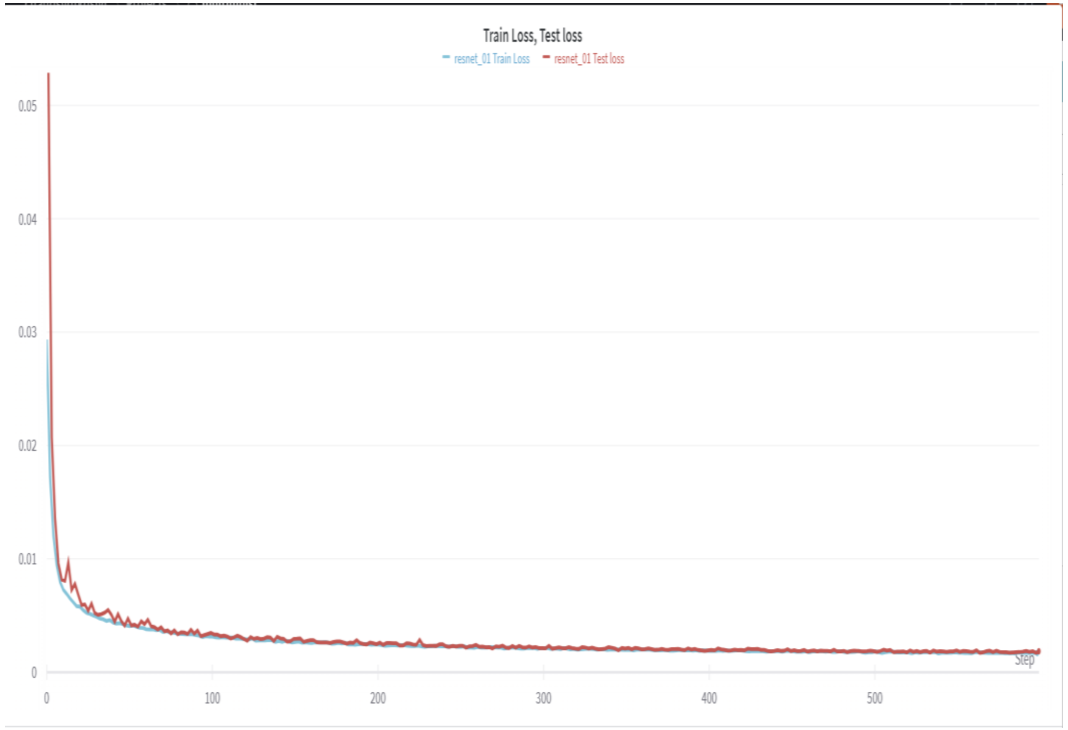
```python
            "Test accuracy" : test_accuracy
        })
        return test_loss, test_accuracy

for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer)
    test_loss, test_accuracy = evaluate(model, test_loader)
    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 %

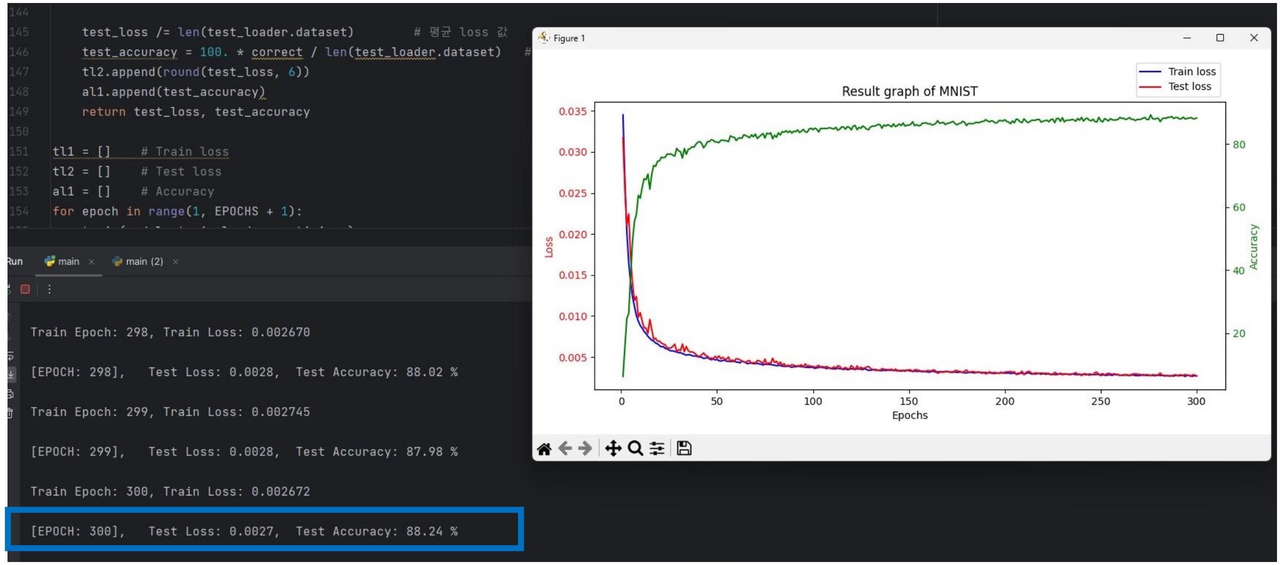




MultiMNIST 1

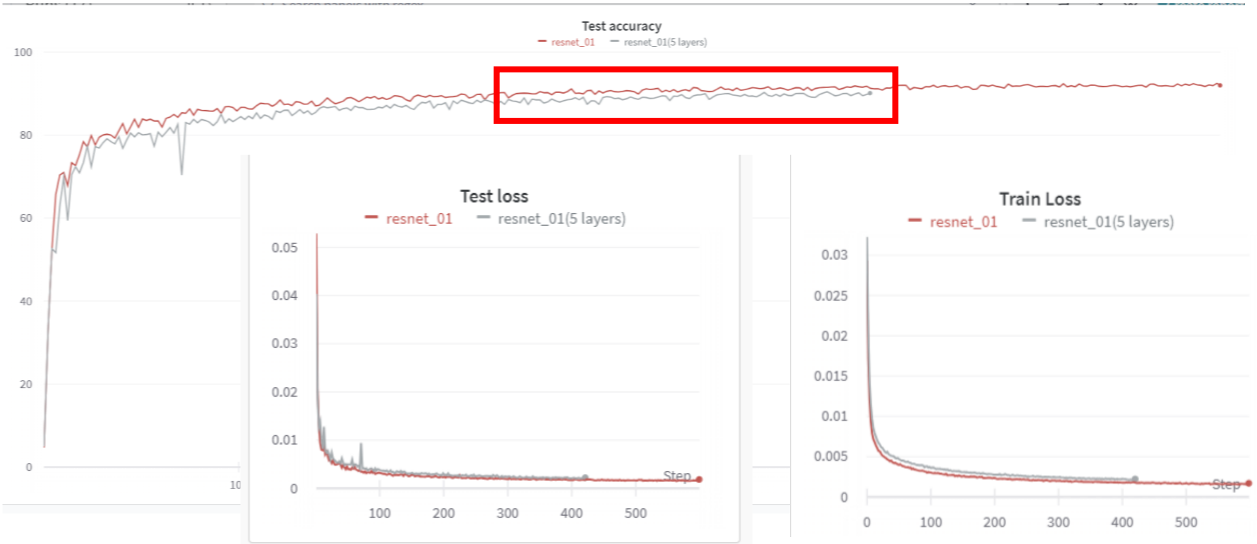
Resnet14

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

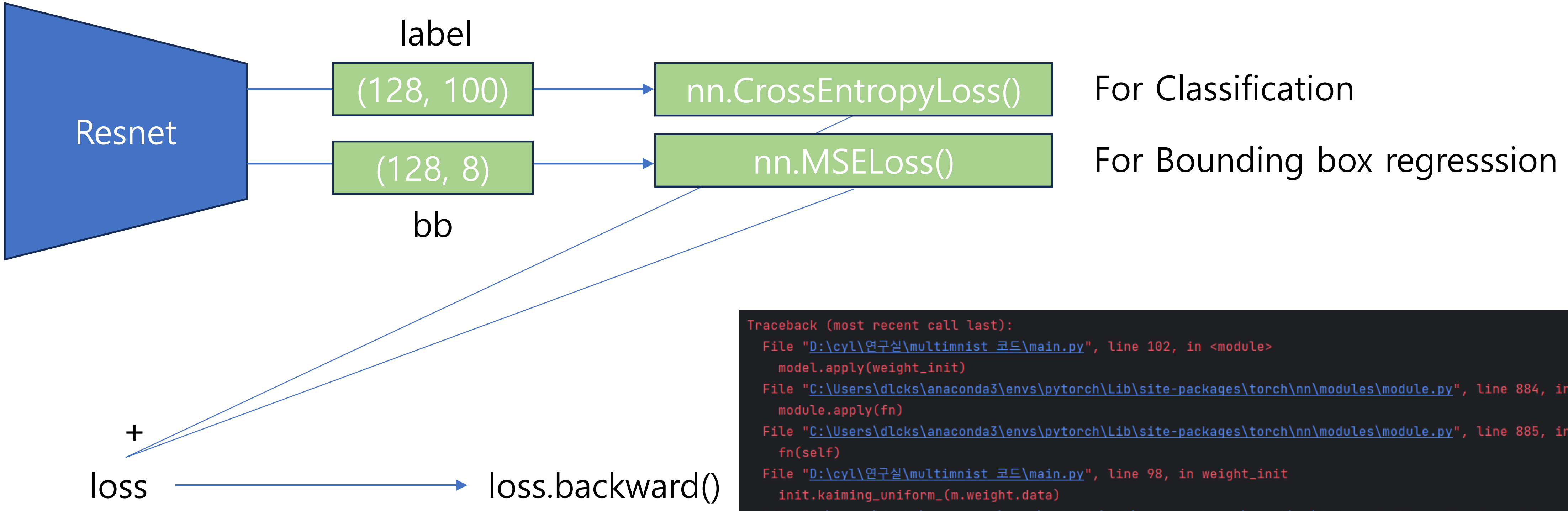


Resnet22

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\dlcks\anaconda3\envs\pytorch\Lib\site-packages\torch\nn\modules\module.py", line 884, in apply
    module.apply(fn)
  File "C:\Users\dlcks\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\dlcks\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.
```