1.
```
import torch
import torch.nn as nn

import torch.optim as optim

import torchvision
import torchvision.transforms as transforms

import warnings
warnings.filterwarnings(action='ignore')
```

```
import visdom

vis = visdom.Visdom()
vis.close(env="main")
```

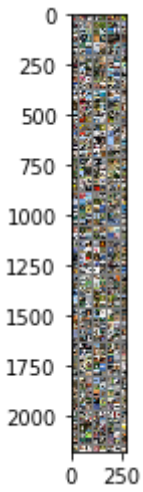
```
def loss_tracker(loss_plot, loss_value, num):
    '''num, loss_value, are Tensor'''
    vis.line(X=num, Y=loss_value, win = loss_plot, update='append')
```

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

torch.manual_seed(777)
if device =='cuda':
    torch.cuda.manual_seed_all(777)
```

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./cifar10', train=True,
                                        download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=512,
shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./cifar10', train=False,
                                       download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse',
'ship', 'truck')
```

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


imshow(torchvision.utils.make_grid(images))


print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```




```
truck   dog horse truck
```

(??)

```
import torchvision.models.vgg as vgg
```

```
cfg =
[32,32,'M',64,64,'M',128,128,128,128,'M',256,256,256,256,'M',512,512,512,512
,'M']
```

```
class VGG(nn.Module):

    def __init__(self, features, num_classes=1000, init_weights=True):
        super(VGG, self).__init__()
        self.features = features
        self.classifier = nn.Sequential(
            nn.Linear(512 * 1 * 1, 4096), #맥스풀링 5번 32 => 1
            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):
        x = self.features(x)
        x = x.view(x.size(0), -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)
```

```
vgg19= VGG(vgg.make_layers(cfg),10,True).to(device)
```

```
a=torch.Tensor(1,3,32,32).to(device)
out = vgg19(a)
```

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(vgg19.parameters(), lr = 0.005,momentum=0.9)
#lr_sche 있었는데 epoch 2번 돌 거라 필요없음
```

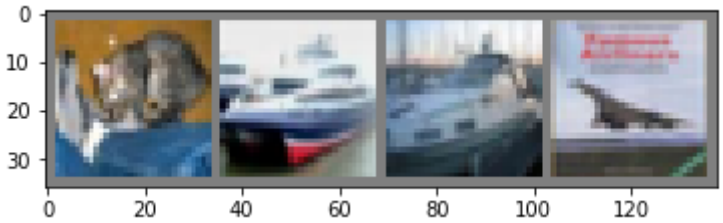
```
loss_plt =
vis.line(Y=torch.Tensor(1).zero_(),opts=dict(title='loss_tracker', legend=
['loss'], showlegend=True))
```

```
print(len(trainloader))
epochs = 2
for epoch in range(epochs):
    running_loss = 0.0
    #lr_sche.step()
    for i, data in enumerate(trainloader, 0):
        # get tue inputs
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = vgg19(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 30 ==29:
```

```
            print('[%d, %5d] loss: %.5f' % (epoch + 1, i + 1,
running_loss/30))
            running_loss = 0.0

print('Finished Training')
```

```
98

[1,    30] loss: 2.30261
[1,    60] loss: 2.30258
[1,    90] loss: 2.30260
[2,    30] loss: 2.30250
[2,    60] loss: 2.30252
[2,    90] loss: 2.30247
Finished Training
```

```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in
range(4)))
```




```
GroundTruth:    cat  ship  ship plane
```

```
outputs = vgg19(images.to(device))
```

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:    dog   car   dog  bird
```

잘 못함..

2.
```
import torch
import torch.nn as nn

import torch.optim as optim

import torchvision
import torchvision.transforms as transforms
```

```
import visdom

vis = visdom.Visdom()
vis.close(env="main")
```

```
def value_tracker(value_plot, value, num):
    '''num, loss_value, are Tensor'''
    vis.line(X=num,
             Y=value,
             win = value_plot,
             update='append')
```

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

torch.manual_seed(777)
if device =='cuda':
    torch.cuda.manual_seed_all(777)
```

```
transform = transforms.Compose([
    transforms.ToTensor()
])

trainset = torchvision.datasets.CIFAR10(root='./cifar10', train=True,
download=True, transform=transform)

print(trainset.data.shape)

train_data_mean = trainset.data.mean( axis=(0,1,2) )
train_data_std = trainset.data.std( axis=(0,1,2) )


print(train_data_mean)
print(train_data_std)

train_data_mean = train_data_mean / 255
train_data_std = train_data_std / 255

print(train_data_mean)
print(train_data_std)
```

```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(train_data_mean, train_data_std)])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(train_data_mean, train_data_std)
])

trainset = torchvision.datasets.CIFAR10(root='./cifar10', train=True,
                                        download=True,
transform=transform_train)
```

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=256,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./cifar10', train=False,
                                       download=True,
transform=transform_test)

testloader = torch.utils.data.DataLoader(testset, batch_size=256,
                                         shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse',
'ship', 'truck')
```

```
import torchvision.models.resnet as resnet
```

```
BasicBlock= resnet.BasicBlock

conv1x1 = resnet.conv1x1
Bottleneck = resnet.Bottleneck
```

```
class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=1000,
zero_init_residual=False):
        super(ResNet, self).__init__()
        self.inplanes = 16
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1,
padding=1,bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.relu = nn.ReLU(inplace=True)


        self.layer1 = self._make_layer(block, 16, layers[0], stride=1)
        self.layer2 = self._make_layer(block, 32, layers[1], stride=1)
        self.layer3 = self._make_layer(block, 64, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 128, layers[3], stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(128 * block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)


        if zero_init_residual:
            for m in self.modules():
                if isinstance(m, Bottleneck):
                    nn.init.constant_(m.bn3.weight, 0)
                elif isinstance(m, BasicBlock):
```

```
                    nn.init.constant_(m.bn2.weight, 0)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride),
                nn.BatchNorm2d(planes * block.expansion),
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes * block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

```
resnet34 = ResNet(resnet.BasicBlock, [3, 4, 6, 3], 10, True).to(device)
# 2*(3+4+6+3) +1(conv1) +1(fc) = 34
```

```
a=torch.Tensor(1,3,32,32).to(device)
out = resnet34(a)
print(out)
```

```
tensor([[-0.0196, -0.0132, -0.0700,  0.0751,  0.0726, -0.0021,  0.0627,
-0.0392,
          0.0873, -0.0145]], grad_fn=<AddmmBackward>)
```

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(resnet34.parameters(), lr = 0.1, momentum = 0.9,
weight_decay = 5e-4)
# 여기서도 lr_sche 빼 줌
```

```
loss_plt =
vis.line(Y=torch.Tensor(1).zero_(),opts=dict(title='loss_tracker', legend=
['loss'], showlegend=True))
acc_plt = vis.line(Y=torch.Tensor(1).zero_(),opts=dict(title='Accuracy',
legend=['Acc'], showlegend=True))
```

```
print(len(trainloader))
epochs = 2

for epoch in range(epochs):  # loop over the dataset multiple times

    running_loss = 0.0

    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = resnet34(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 30 == 29:    # print every 30 mini-batches
            value_tracker(loss_plt, torch.Tensor([running_loss/30]),
torch.Tensor([i + epoch*len(trainloader) ]))
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 30))
            running_loss = 0.0


print('Finished Training')
```

```
196
[1,    30] loss: 1.902
[1,    60] loss: 1.784
[1,    90] loss: 1.685
[1,   120] loss: 1.630
[1,   150] loss: 1.561
[1,   180] loss: 1.483
[2,    30] loss: 1.381
[2,    60] loss: 1.267
[2,    90] loss: 1.234
[2,   120] loss: 1.189
[2,   150] loss: 1.133
[2,   180] loss: 1.060
Finished Training
```

끝