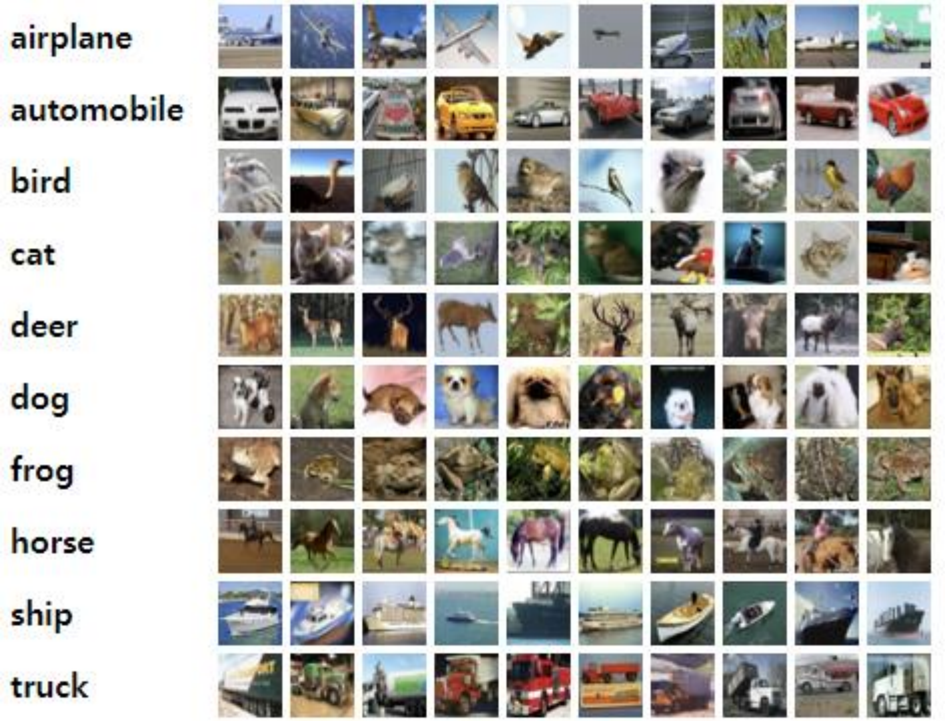
StandAlone

DeepLearning

김영민

1. CIFAR-10

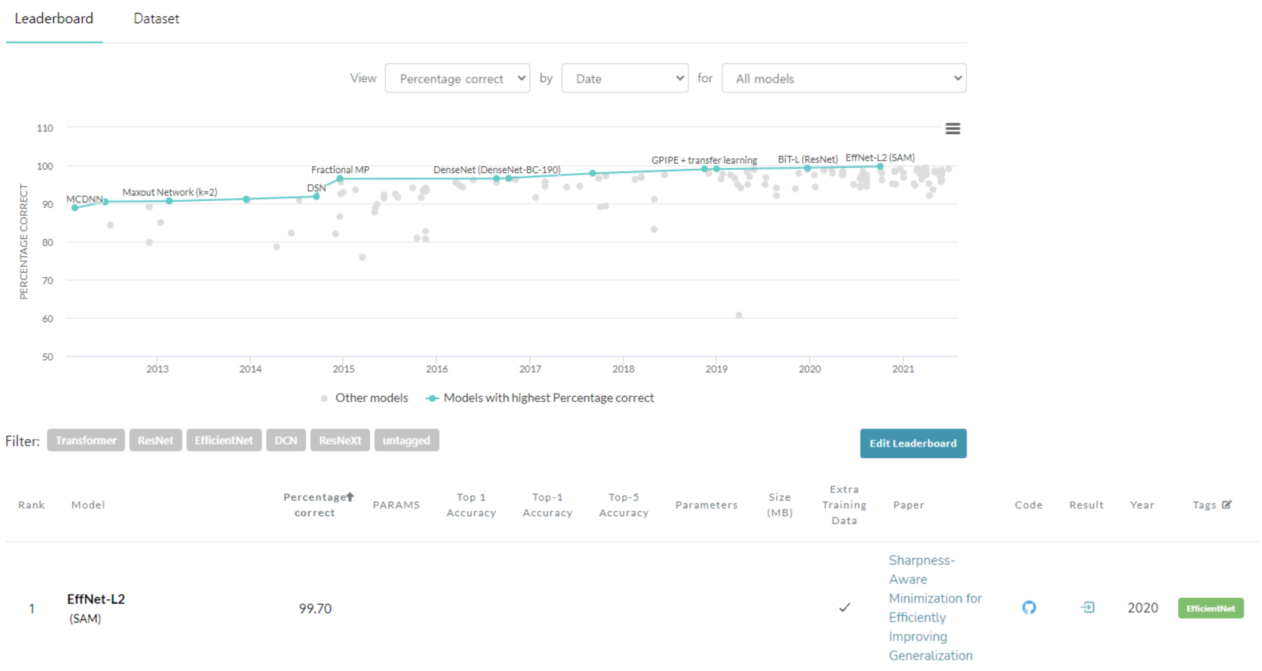
Dataset



Train : 60,000

Test : 10,000

SOTA



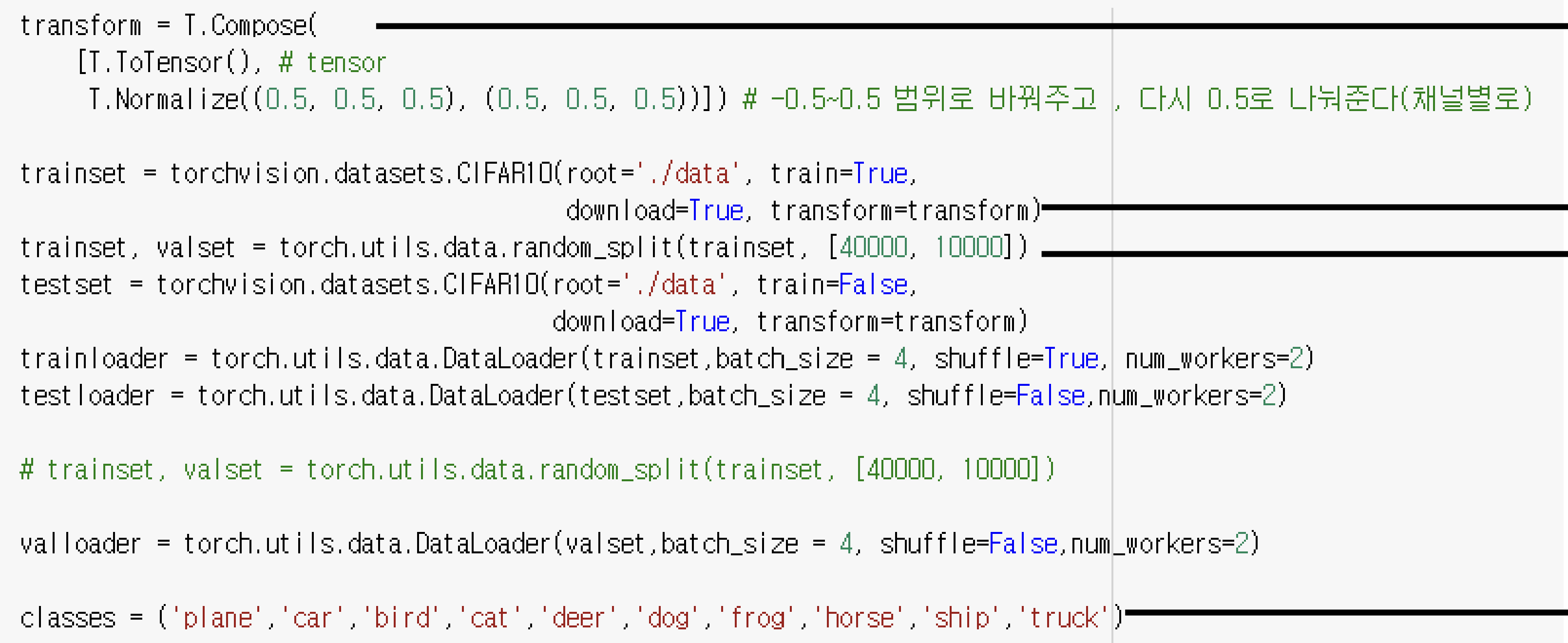
View: Percentage correct by: Date for: All models

Rank	Model	Percentage correct	PARAMS	Top 1 Accuracy	Top-1 Accuracy	Top-5 Accuracy	Parameters	Size (MB)	Extra Training Data	Paper	Code	Result	Year	Tags
1	EffNet-L2 (SAM)	99.70							✓	Sharpness-Aware Minimization for Efficiently Improving Generalization			2020	EfficientNet

2021 SOTA model = EffNet-L2

2. Deep Learning Code

Data Loading



```python
transform = T.Compose(
    [T.ToTensor(), # tensor
     T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]) # -0.5~0.5 범위로 바꿔주고 , 다시 0.5로 나눠준다(채널별로)

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000])
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,batch_size = 4, shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset,batch_size = 4, shuffle=False,num_workers=2)

# trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000])

valloader = torch.utils.data.DataLoader(valset,batch_size = 4, shuffle=False,num_workers=2)

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

Data Transformation

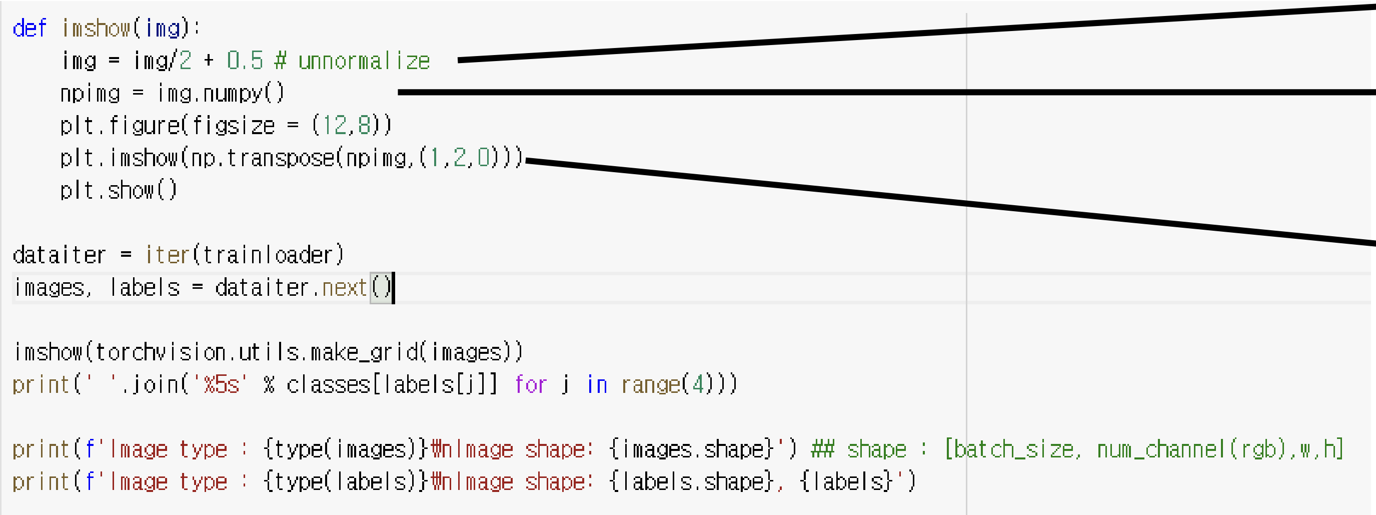
Data Download

Data Split

Define Class

2. Deep Learning Code

Visualization

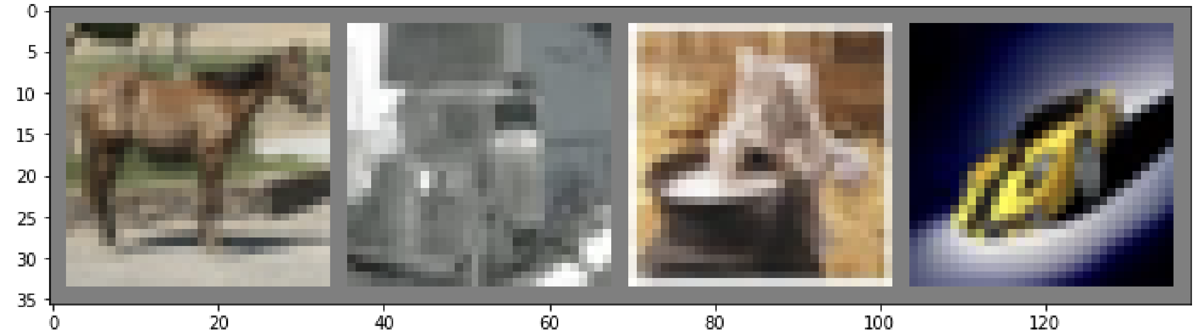


```
def imshow(img):
    img = img/2 + 0.5 # unnormalize
    npimg = img.numpy()
    plt.figure(figsize = (12,8))
    plt.imshow(np.transpose(npimg,(1,2,0)))
    plt.show()

dataiter = iter(trainloader)
images, labels = dataiter.next()

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

print(f'Image type : {type(images)}₩nImage shape: {images.shape}') ## shape : [batch_size, num_channel(rgb),w,h]
print(f'Image type : {type(labels)}₩nImage shape: {labels.shape}, {labels}')
```

Transformation -> Original

Tensor -> numpy

Tensor = [C,H,W]
-> plot imshow shape = [H,W,C]

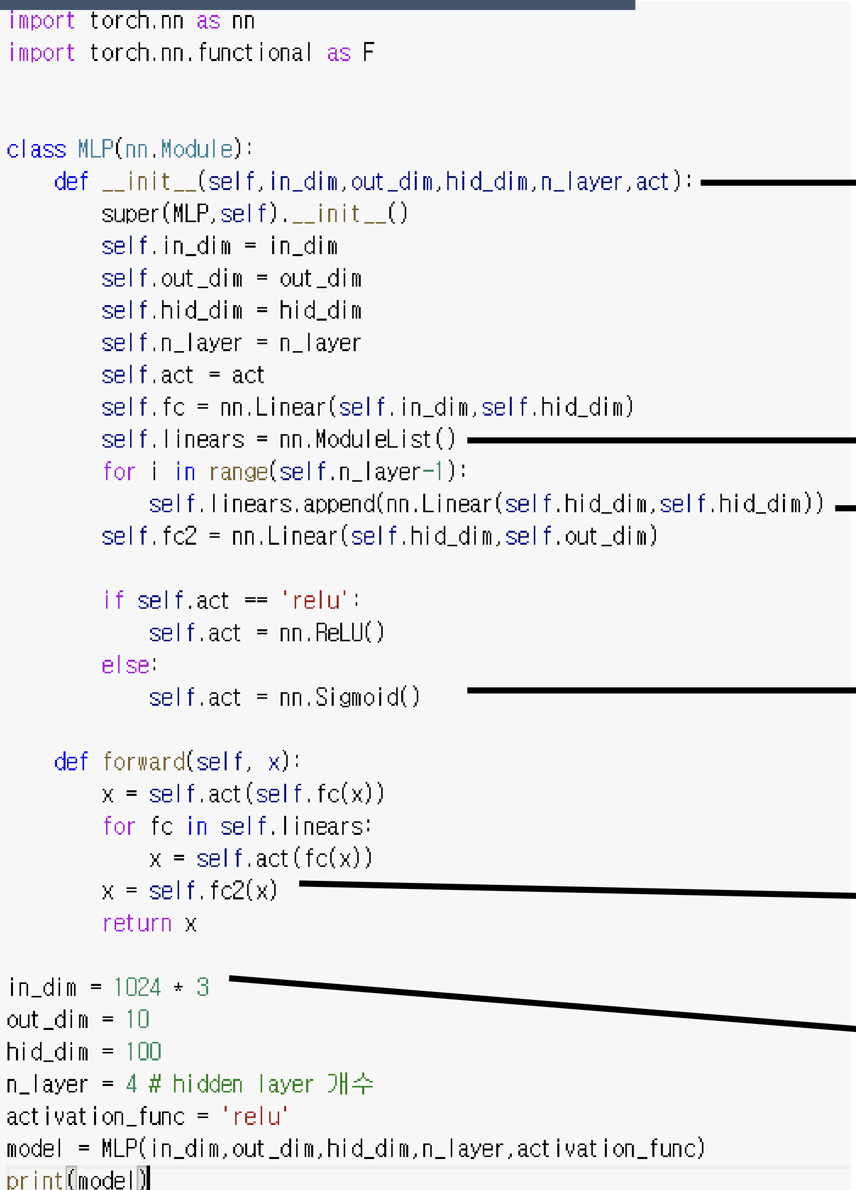


```
horse truck   cat    car
Image type : <class 'torch.Tensor'>
Image shape: torch.Size([4, 3, 32, 32])
Image type : <class 'torch.Tensor'>
Image shape: torch.Size([4]), tensor([7, 9, 3, 1])
```

[batch_size,C,H,W]

2. Deep Learning Code

Network Architecture



```python
import torch.nn as nn
import torch.nn.functional as F


class MLP(nn.Module):
    def __init__(self,in_dim,out_dim,hid_dim,n_layer,act):
        super(MLP,self).__init__()
        self.in_dim = in_dim
        self.out_dim = out_dim
        self.hid_dim = hid_dim
        self.n_layer = n_layer
        self.act = act
        self.fc = nn.Linear(self.in_dim,self.hid_dim)
        self.linears = nn.ModuleList()
        for i in range(self.n_layer-1):
            self.linears.append(nn.Linear(self.hid_dim,self.hid_dim))
        self.fc2 = nn.Linear(self.hid_dim,self.out_dim)

        if self.act == 'relu':
            self.act = nn.ReLU()
        else:
            self.act = nn.Sigmoid()

    def forward(self, x):
        x = self.act(self.fc(x))
        for fc in self.linears:
            x = self.act(fc(x))
        x = self.fc2(x)
        return x

in_dim = 1024 * 3
out_dim = 10
hid_dim = 100
n_layer = 4 # hidden layer 개수
activation_func = 'relu'
model = MLP(in_dim,out_dim,hid_dim,n_layer,activation_func)
print(model)
```

In_dim = input 값(이미지 하나 크기)

out_dim = output 개수(class 가 총 몇개인지)

Hid_dim = hidden layer의 노드 개수

N_layer = hidden layer 개수

Act = Activation Function 종류

변수 정의

Module List

Layer 개수에 따라 Linear Module 추가

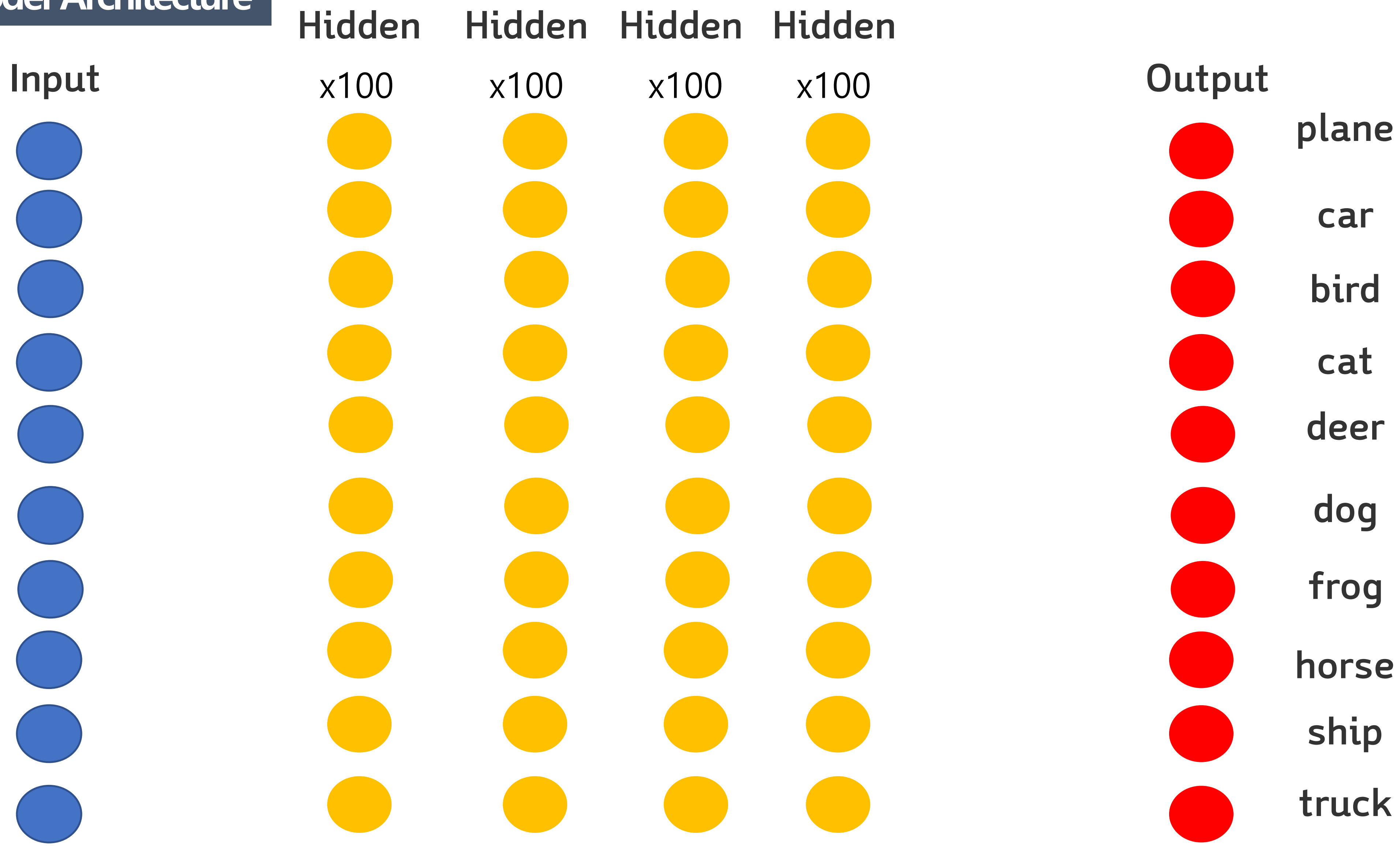
Activation Function

마지막 층에는 Activation Function 적용 X

32*32*3(RGB)

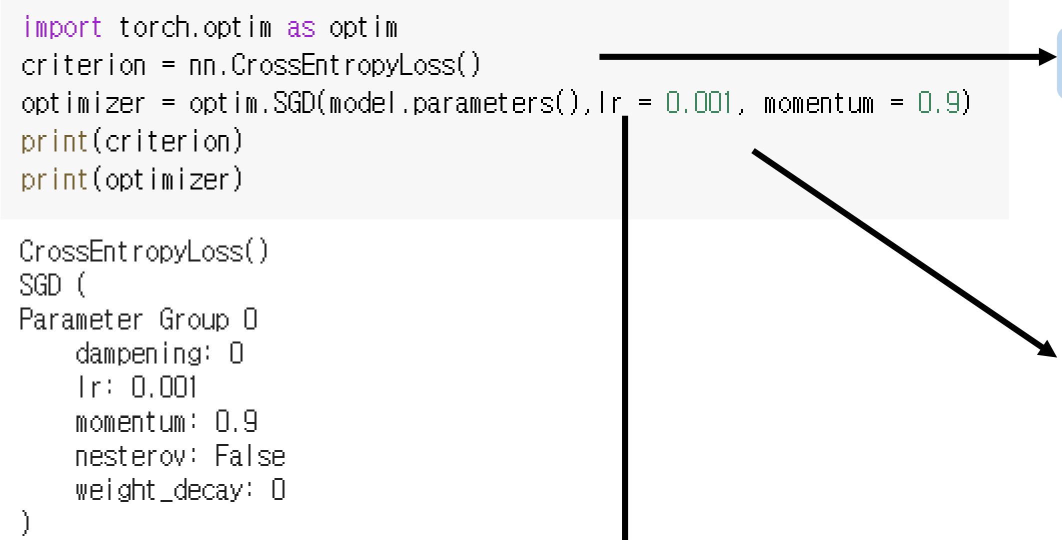
2. Deep Learning Code

Visualize Model Architecture



2. Deep Learning Code

Optimizer & Loss Function



```python
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),lr = 0.001, momentum = 0.9)
print(criterion)
print(optimizer)
```

```
CrossEntropyLoss()
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.001
    momentum: 0.9
    nesterov: False
    weight_decay: 0
)
```

https://pytorch.org/docs/stable/nn.html#loss-functions

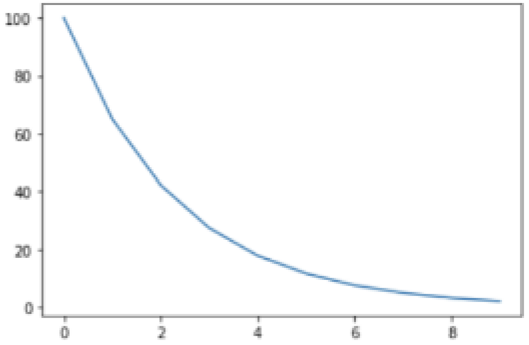
Loss CrossEntropy, L1 Loss, MSE, BCE Loss..

Optimizer SGD, Adam, AdamW..

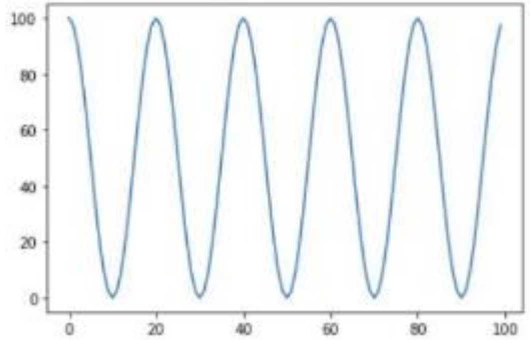
LR scheduler

optim.lr_scheduler

Lambda LR

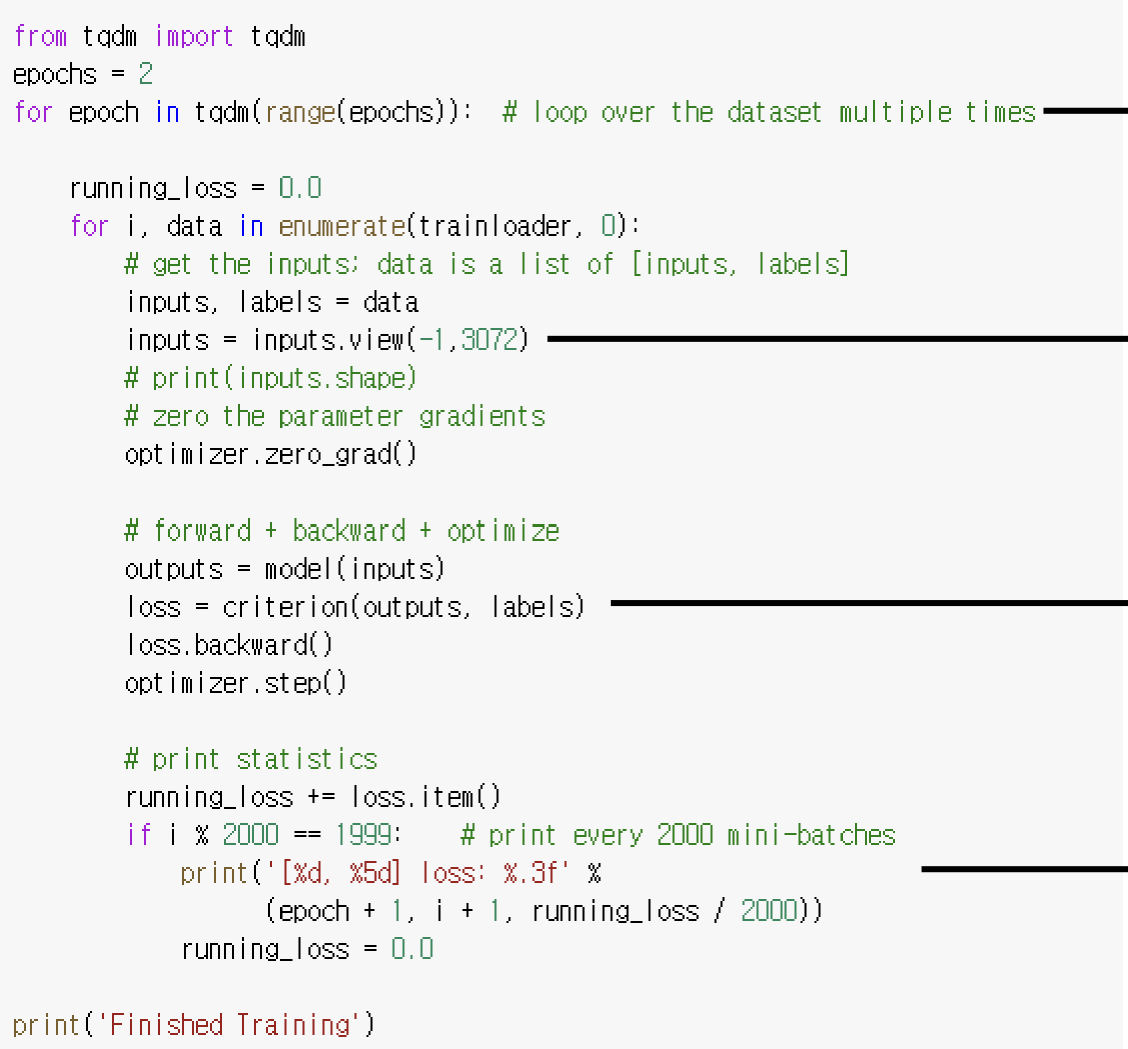


CosAnnealing LR

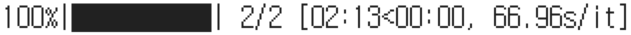


2. Deep Learning Code

Train Code



```python
from tqdm import tqdm
epochs = 2
for epoch in tqdm(range(epochs)):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.view(-1,3072)
        # print(inputs.shape)
        # zero the parameter gradients
        optimizer.zero_grad()

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

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```



```
100%|██████████| 2/2 [02:13<00:00, 66.96s/it]
```

Tqdm = 상태바 사용

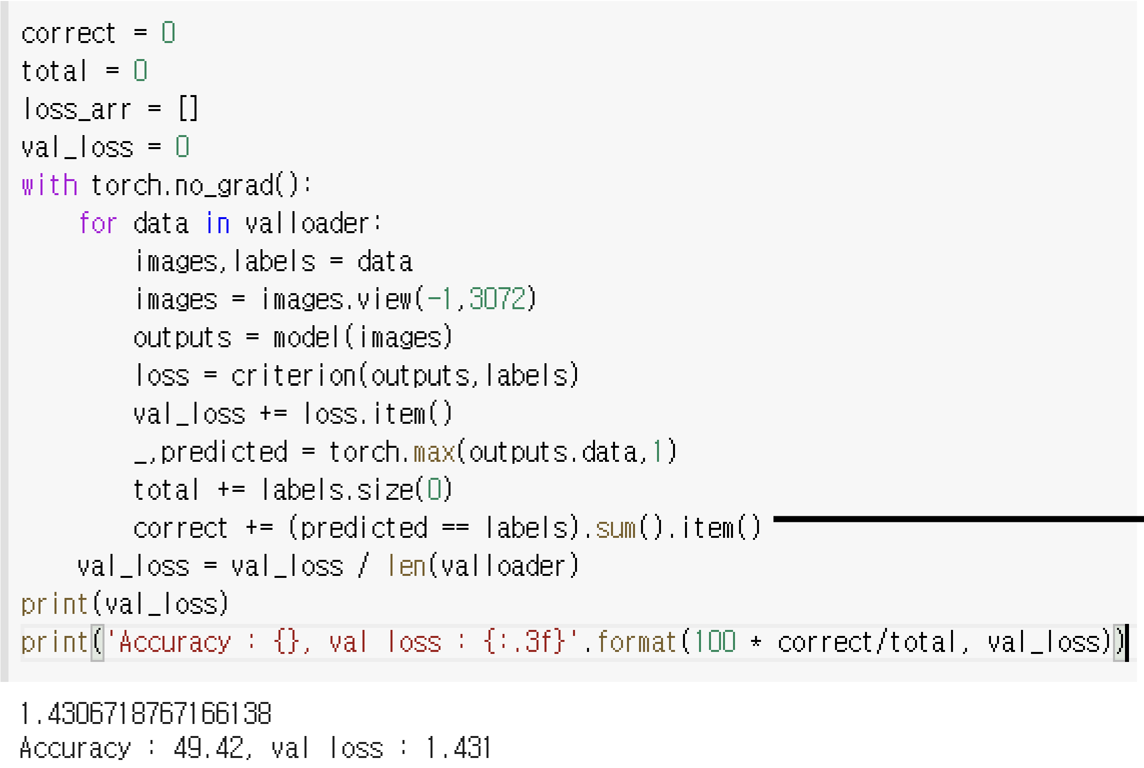
2차원으로 shape 변경

Loss 계산

Iteration 2000번 돌 때 마다 평균 loss 추출

2. Deep Learning Code

Test Code

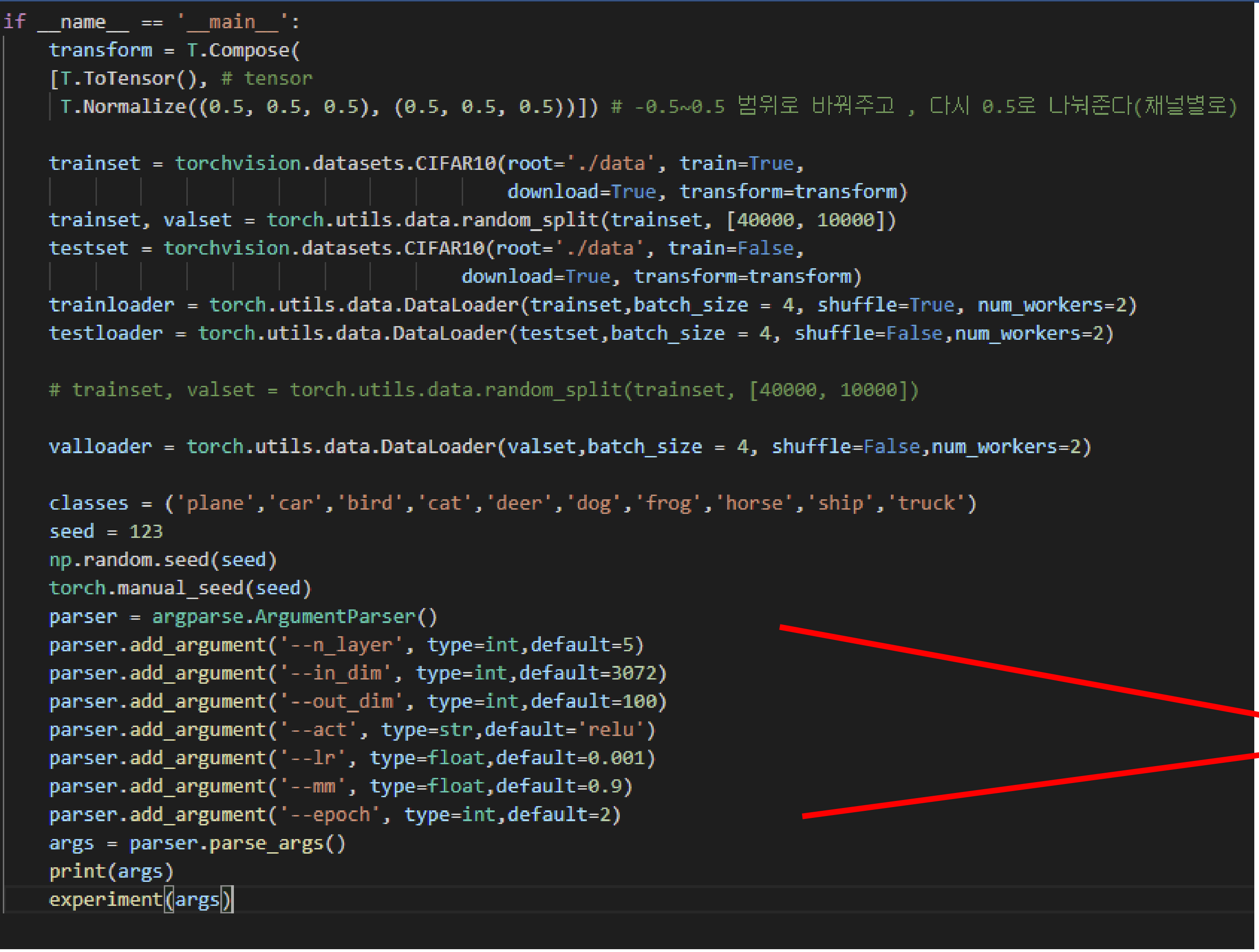


```
correct = 0
total = 0
loss_arr = []
val_loss = 0
with torch.no_grad():
    for data in valloader:
        images,labels = data
        images = images.view(-1,3072)
        outputs = model(images)
        loss = criterion(outputs,labels)
        val_loss += loss.item()
        _,predicted = torch.max(outputs.data,1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    val_loss = val_loss / len(valloader)
print(val_loss)
print('Accuracy : {}, val loss : {:.3f}'.format(100 * correct/total, val_loss))
```

```
1.4306718767166138
Accuracy : 49.42, val loss : 1.431
```

Iteration 돌 때 마다 accuracy 계산

2. Deep Learning Code

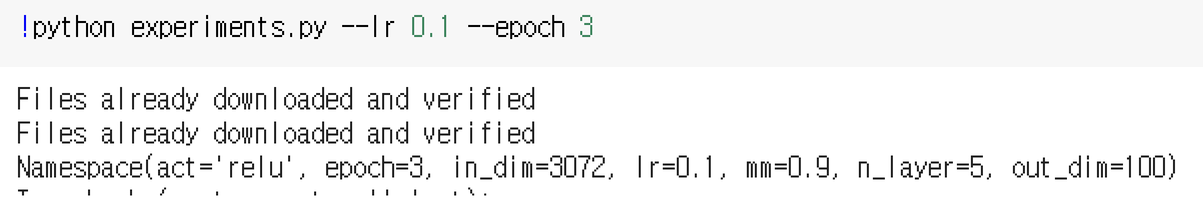


```python
if __name__ == '__main__':
    transform = T.Compose(
    [T.ToTensor(), # tensor
     T.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]) # -0.5~0.5 범위로 바꿔주고 , 다시 0.5로 나눠준다(채널별로)

    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
    trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000])
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                        download=True, transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset,batch_size = 4, shuffle=True, num_workers=2)
    testloader = torch.utils.data.DataLoader(testset,batch_size = 4, shuffle=False,num_workers=2)

    # trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000])

    valloader = torch.utils.data.DataLoader(valset,batch_size = 4, shuffle=False,num_workers=2)

    classes = ('plane','car','bird','cat','deer','dog','frog','horse','ship','truck')
    seed = 123
    np.random.seed(seed)
    torch.manual_seed(seed)
    parser = argparse.ArgumentParser()
    parser.add_argument('--n_layer', type=int,default=5)
    parser.add_argument('--in_dim', type=int,default=3072)
    parser.add_argument('--out_dim', type=int,default=100)
    parser.add_argument('--act', type=str,default='relu')
    parser.add_argument('--lr', type=float,default=0.001)
    parser.add_argument('--mm', type=float,default=0.9)
    parser.add_argument('--epoch', type=int,default=2)
    args = parser.parse_args()
    print(args)
    experiment(args)
```

Argparser 문 정의

Experiments.py

사용 예시

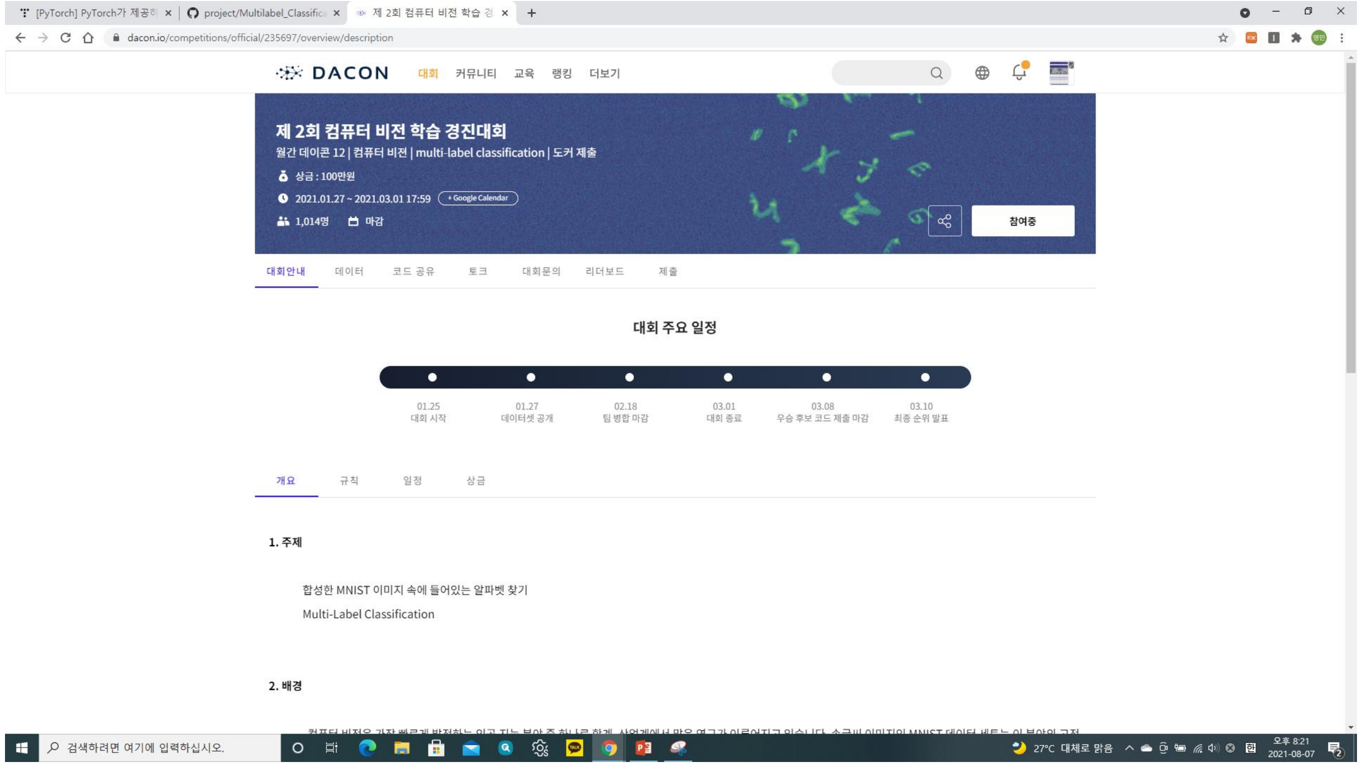


```
!python experiments.py --lr 0.1 --epoch 3
```

```
Files already downloaded and verified
Files already downloaded and verified
Namespace(act='relu', epoch=3, in_dim=3072, lr=0.1, mm=0.9, n_layer=5, out_dim=100)
```

3. EMNIST

Introduce Contest


[PyTorch] PyTorch가 제공하
project/Multilabel_Classifica
제 2회 컴퓨터 비전 학습 경
dacon.io/competitions/official/235697/overview/description
DACON
대회
커뮤니티
교육
랭킹
더보기
참여중
대회안내
데이터
코드 공유
토크
대회문의
리더보드
제출
개요
규칙
일정
상금
검색하려면 여기에 입력하십시오.
27°C 대체로 맑음
오후 8:21
2021-08-07

제 2회 컴퓨터 비전 학습 경진대회

월간 데이콘 12 | 컴퓨터 비전 | multi-label classification | 도커 제출

상금 : 100만원

2021.01.27 ~ 2021.03.01 17:59 + Google Calendar

1,014명 마감

대회 주요 일정

01.25	01.27	02.18	03.01	03.08	03.10
대회 시작	데이터셋 공개	팀 병합 마감	대회 종료	우승 후보 코드 제출 마감	최종 순위 발표

1. 주제

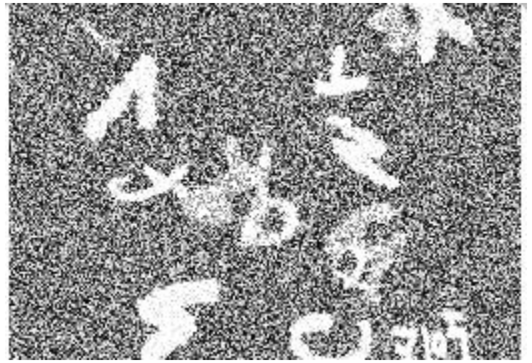
합성한 MNIST 이미지 속에 들어있는 알파벳 찾기

Multi-Label Classification

2. 배경

컴퓨터 비전은 가장 빠르게 발전하는 인공 지능 분야 중 하나로 학계, 산업계에서 많은 연구가 이루어지고 있습니다. 손글씨 이미지인 MNIST 데이터 세트는 이 분야의 고전

Multi-label Classification



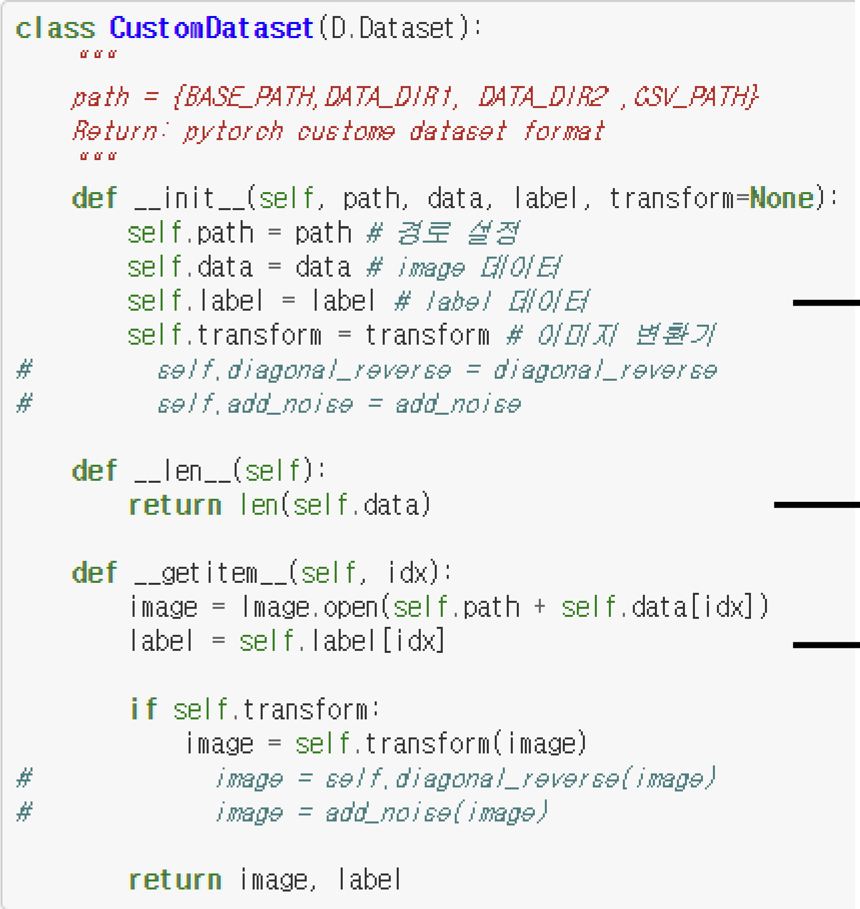
A-Z 글자 무작위 배치

+

Noise

3. EMNIST

Data Loading



```
class CustomDataset(D.Dataset):
    """
    path = {BASE_PATH,DATA_DIR1, DATA_DIR2 ,CSV_PATH}
    Return: pytorch custome dataset format
    """
    def __init__(self, path, data, label, transform=None):
        self.path = path # 경로 설정
        self.data = data # image 데이터
        self.label = label # label 데이터
        self.transform = transform # 이미지 변환기
#         self.diagonal_reverse = diagonal_reverse
#         self.add_noise = add_noise

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image = Image.open(self.path + self.data[idx])
        label = self.label[idx]

        if self.transform:
            image = self.transform(image)
#             image = self.diagonal_reverse(image)
#             image = add_noise(image)

        return image, label
```

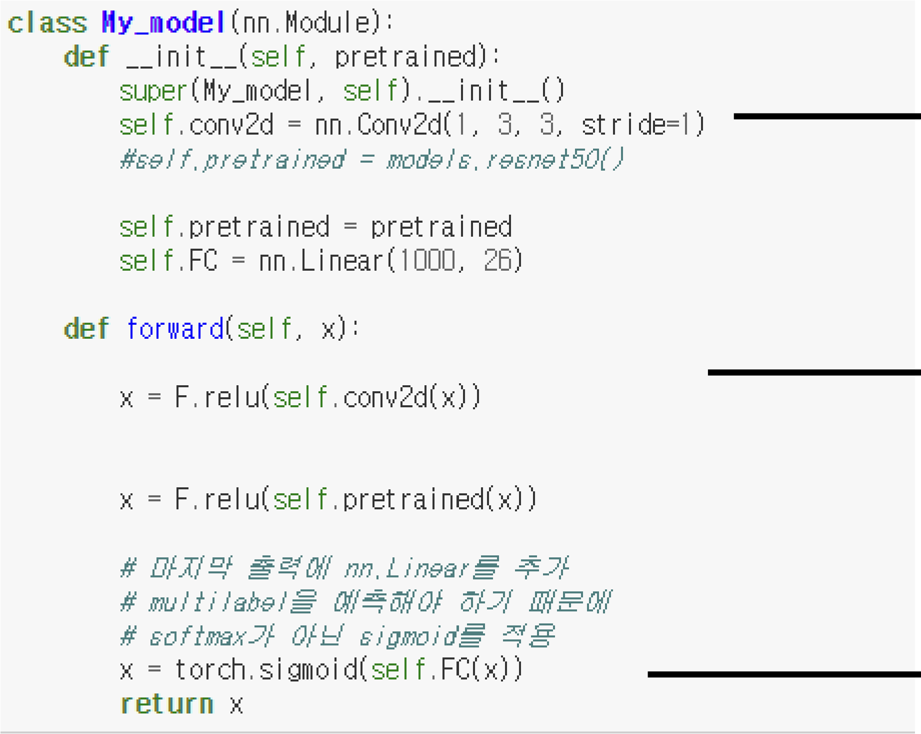
Data 초기 변수 설정 :
경로, data, label, Transformation

Data 개수

Transformation 적용

3. EMNIST

Model Architecture



```
class My_model(nn.Module):
    def __init__(self, pretrained):
        super(My_model, self).__init__()
        self.conv2d = nn.Conv2d(1, 3, 3, stride=1)
        #self.pretrained = models.resnet50()

        self.pretrained = pretrained
        self.FC = nn.Linear(1000, 26)

    def forward(self, x):

        x = F.relu(self.conv2d(x))


        x = F.relu(self.pretrained(x))

        # 마지막 출력에 nn.Linear를 추가
        # multilabel을 예측해야 하기 때문에
        # softmax가 아닌 sigmoid를 적용
        x = torch.sigmoid(self.FC(x))
        return x
```

Pretraining 된 모델 적용 = Efficient-Netb3

+

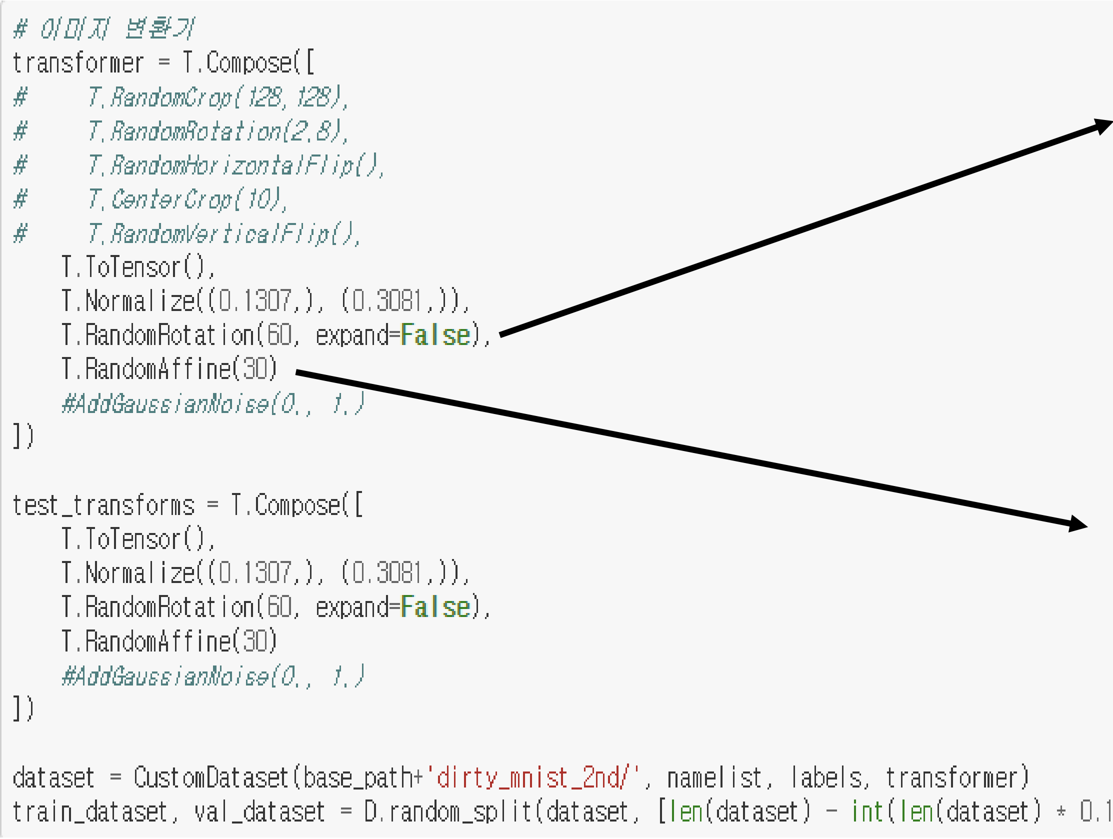
Convolution 통과

이미지 -> Convolution -> ReLU

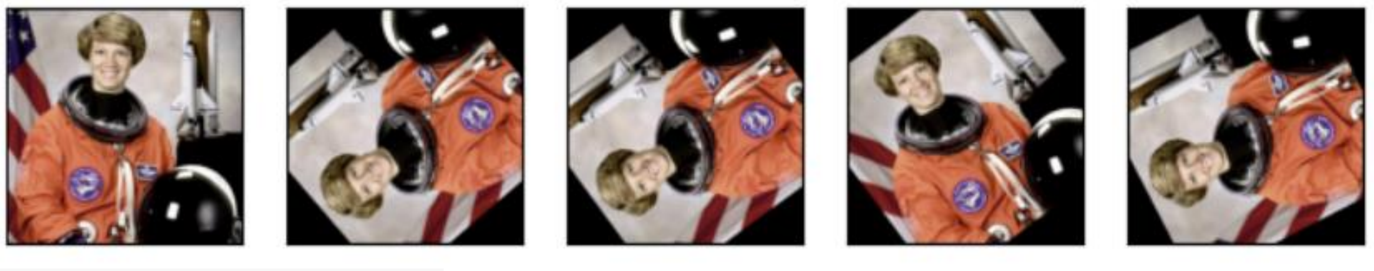
마지막 layer에 Sigmoid 적용

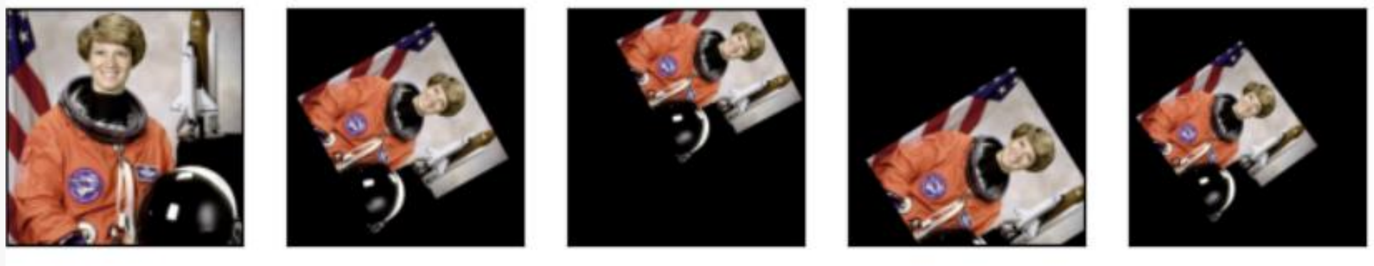
3. EMNIST

Transformation



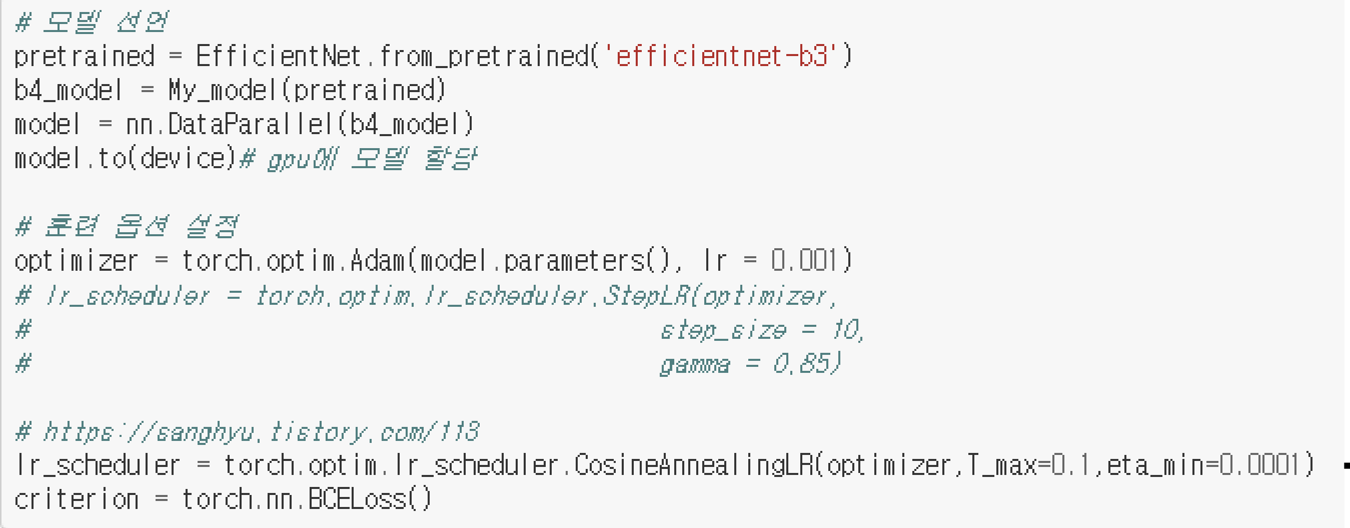
```
# 이미지 변환기
transformer = T.Compose([
#     T.RandomCrop(128,128),
#     T.RandomRotation(2,8),
#     T.RandomHorizontalFlip(),
#     T.CenterCrop(10),
#     T.RandomVerticalFlip(),
    T.ToTensor(),
    T.Normalize((0.1307,), (0.3081,)),
    T.RandomRotation(60, expand=False),
    T.RandomAffine(30)
    #AddGaussianNoise(0., 1.)
])

test_transforms = T.Compose([
    T.ToTensor(),
    T.Normalize((0.1307,), (0.3081,)),
    T.RandomRotation(60, expand=False),
    T.RandomAffine(30)
    #AddGaussianNoise(0., 1.)
])

dataset = CustomDataset(base_path+'dirty_mnist_2nd/', namelist, labels, transformer)
train_dataset, val_dataset = D.random_split(dataset, [len(dataset) - int(len(dataset) * 0.1), int(len(dataset) * 0.1)])
```



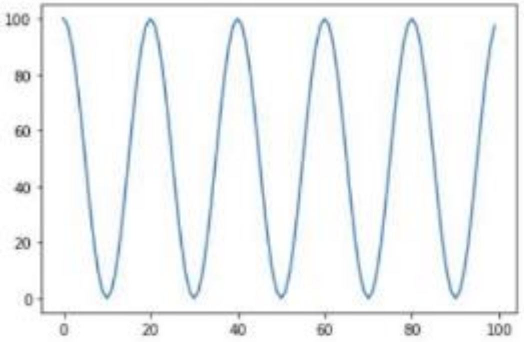


3. EMNIST

Define Hyperparameter



```
# 모델 선언
pretrained = EfficientNet.from_pretrained('efficientnet-b3')
b4_model = My_model(pretrained)
model = nn.DataParallel(b4_model)
model.to(device)# gpu에 모델 할당

# 훈련 옵션 설정
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
# lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
#                                                step_size = 10,
#                                                gamma = 0.85)

# https://sanghyu.tistory.com/113
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,T_max=0.1,eta_min=0.0001)
criterion = torch.nn.BCELoss()
```



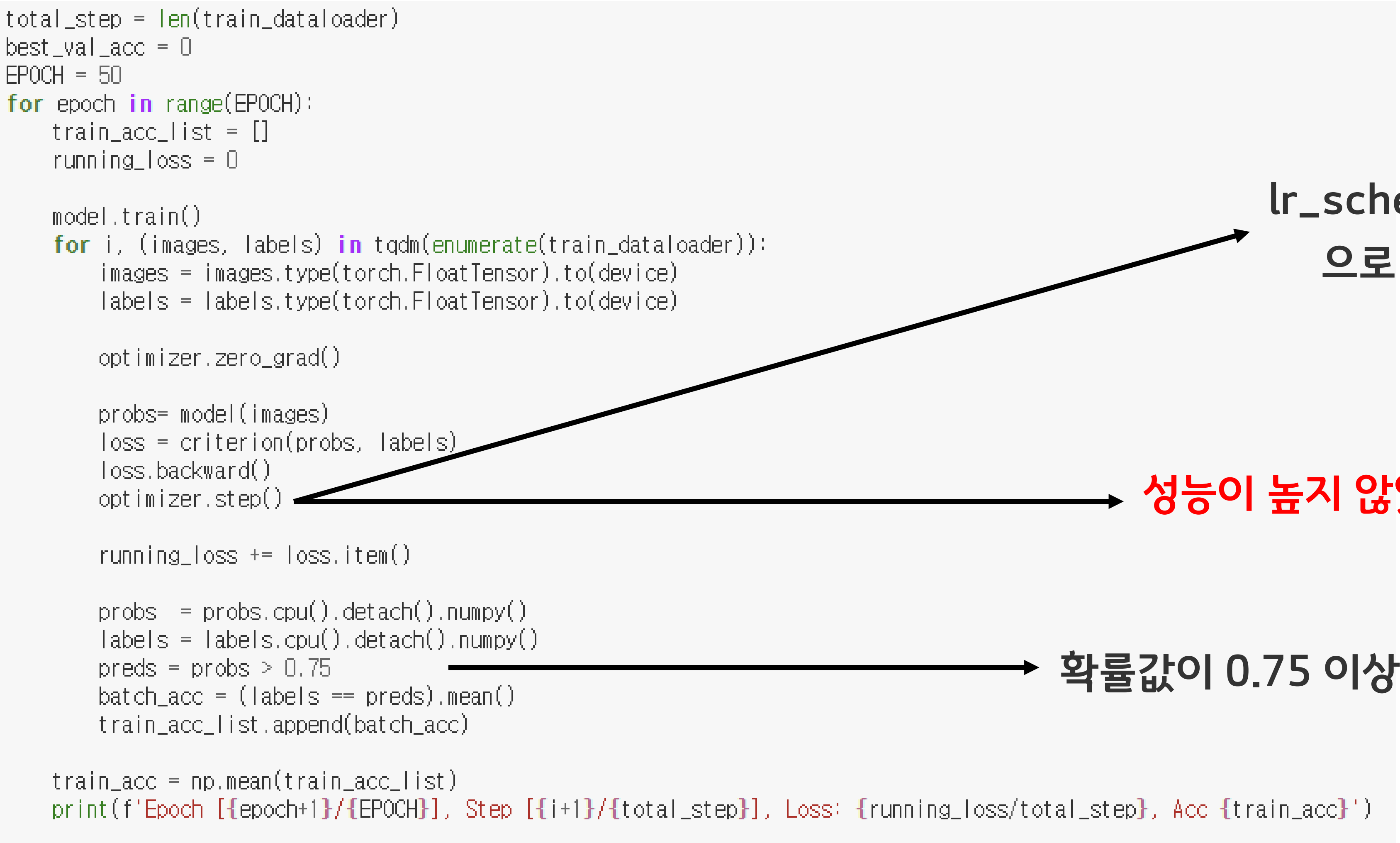
최대 lr = 0.1, 최소 lr = 0.0001

Binary Cross Entropy Loss

성능이 높지 않았던 원인 1

3. EMNIST

Train



```python
total_step = len(train_dataloader)
best_val_acc = 0
EPOCH = 50
for epoch in range(EPOCH):
    train_acc_list = []
    running_loss = 0

    model.train()
    for i, (images, labels) in tqdm(enumerate(train_dataloader)):
        images = images.type(torch.FloatTensor).to(device)
        labels = labels.type(torch.FloatTensor).to(device)

        optimizer.zero_grad()

        probs= model(images)
        loss = criterion(probs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        probs  = probs.cpu().detach().numpy()
        labels = labels.cpu().detach().numpy()
        preds = probs > 0.75
        batch_acc = (labels == preds).mean()
        train_acc_list.append(batch_acc)

    train_acc = np.mean(train_acc_list)
    print(f'Epoch [{epoch+1}/{EPOCH}], Step [{i+1}/{total_step}], Loss: {running_loss/total_step}, Acc {train_acc}')
```

lr_scheduler.step()
으로 변경해야함

성능이 높지 않았던 원인 2

확률값이 0.75 이상이면 예측값이 정답값이라고 판단