1. 데이터 준비하기

MNIST를 이용해서 이미지 변환을 통해서 3채널의 이미지로 만들었습니다

```
def mnist_image_augment(mnist, scale=True, rotate=True, shear=True, colour=True, gaussian=True, invert=True):
    l = len(mnist)

    SC = np.random.normal(1, 0.3, size=l) # scale
    SH = np.random.normal(0, 1, size=(l, 3, 2)) # shear
    R = np.random.normal(0, 20, size=l) # rotate
    C = np.random.randint(22, size=l) # colour
    G = np.random.randint(30, size=l) # noise
    I = np.random.randint(2, size=l) # invert

    augmented = []
```

위와 같이 MNIST data를 scale,rotate,shear,color,gaussian,invert를 통해 변환해서 사용했습니다.

2.모델 구축하기

CNN모델중 VGG-16이 성능이 좋다고 배우게 되어서 한 레이어에 컨볼루션 신경망을 겹쳐서 쌓는 방식으로 아래처럼 모델을 구축해보았습니다.

```
self.layer1=torch.nn.Sequential(
    torch.nn.Conv2d(3,n_channels_1,kernel_size=2,stride=1),
    torch.nn.ReLU(),
    torch.nn.Conv2d(n_channels_1,10,kernel_size=2,stride=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2,stride=2)
)
self.layer2=torch.nn.Sequential(
    torch.nn.Conv2d(10,14,kernel_size=3,stride=1),
    torch.nn.ReLU(),
    torch.nn.Conv2d(14,16,kernel_size=4,stride=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2,stride=2)
)
self.fc3=torch.nn.Linear(4*4*n_channels_2,128,bias=True)
torch.nn.init.xavier_uniform_(self.fc3.weight)

self.layer3=torch.nn.Sequential(
    self.fc3,
    torch.nn.ReLU(),
    torch.nn.Dropout(p=1-self.keep_prob)
)
self.fc4=torch.nn.Linear(128,64,bias=True)
torch.nn.init.xavier_uniform_(self.fc4.weight)

self.layer4=torch.nn.Sequential(
    self.fc4,
    torch.nn.ReLU(),
    torch.nn.Dropout(p=1-self.keep_prob)
)
self.fc5=torch.nn.Linear(64,32,bias=True)
torch.nn.init.xavier_uniform_(self.fc5.weight)

self.layer5=torch.nn.Sequential(
    self.fc5,
    torch.nn.ReLU(),
    torch.nn.Dropout(p=1-self.keep_prob)
)
self.fc6=torch.nn.Linear(32,10,bias=True)
torch.nn.init.xavier_uniform_(self.fc6.weight)
```

Epoch을 5번으로 했을 때 성능이 아래와 같았고, 이 모델의 성능이 별로인 이유가 컨볼루션 신경망이 너무 겹쳐져 있어 모델을 간소화 해야할 것 같다고 생각했습니다.

```
[Epoch:    1] cost = 2.16945669
Dev
data_loader: 100%|████████████████████████████████████████| 313/313 [00:03<00:00, 96.78it/s]
train_loader:   0%|                                        | 2/1563 [00:00<01:18, 19.81it/s]
Accuracy:0.3698()
train_loader: 100%|████████████████████████████████████████| 1563/1563 [01:12<00:00, 21.68it/s]
data_loader:   2%|▉                                       | 5/313 [00:00<00:06, 44.27it/s]
[Epoch:    2] cost = 1.84154298
Dev
data_loader: 100%|████████████████████████████████████████| 313/313 [00:03<00:00, 96.48it/s]
train_loader:   0%|                                        | 2/1563 [00:00<01:20, 19.43it/s]
Accuracy:0.4626()
train_loader: 100%|████████████████████████████████████████| 1563/1563 [01:11<00:00, 21.80it/s]
data_loader:   2%|▉                                       | 7/313 [00:00<00:04, 61.44it/s]
[Epoch:    3] cost = 1.67250609
Dev
data_loader: 100%|████████████████████████████████████████| 313/313 [00:03<00:00, 98.48it/s]
train_loader:   0%|                                        | 2/1563 [00:00<01:24, 18.53it/s]
Accuracy:0.5237()
train_loader: 100%|████████████████████████████████████████| 1563/1563 [01:09<00:00, 22.53it/s]
data_loader:   2%|▉                                       | 6/313 [00:00<00:05, 58.86it/s]
[Epoch:    4] cost = 1.56713676
Dev
data_loader: 100%|████████████████████████████████████████| 313/313 [00:03<00:00, 98.33it/s]
train_loader:   0%|                                        | 2/1563 [00:00<01:28, 17.55it/s]
Accuracy:0.5633()
train_loader: 100%|████████████████████████████████████████| 1563/1563 [01:09<00:00, 22.46it/s]
data_loader:   2%|▉                                       | 7/313 [00:00<00:04, 66.08it/s]
[Epoch:    5] cost = 1.51240713
Dev
data_loader: 100%|████████████████████████████████████████| 313/313 [00:03<00:00, 97.72it/s]
Accuracy:0.5735()
```

모델을 아래처럼 컨볼루션 신경망을 2개만 사용해서 구축한 뒤 epoch을 늘려가며 train 시켰을때, 훨씬 더 좋은 성능을 보였습니다.

```
class CNN(torch.nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.in_dim=28*28*3
        self.out_dim=10

        n_channels_1=6
        n_channels_2=16

        self.layer1=torch.nn.Sequential(
            torch.nn.Conv2d(3,n_channels_1,kernel_size=5,stride=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2,stride=2)
        )
        self.layer2=torch.nn.Sequential(
            torch.nn.Conv2d(n_channels_1,n_channels_2,kernel_size=5,stride=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2,stride=2)
        )
        self.fc3=torch.nn.Linear(4*4*n_channels_2,120,bias=True)
        torch.nn.init.xavier_uniform_(self.fc3.weight)

        self.layer3=torch.nn.Sequential(
            self.fc3,
            torch.nn.ReLU(),
            torch.nn.Dropout(p=1-self.keep_prob)
        )
        self.fc4=torch.nn.Linear(120,80,bias=True)
        torch.nn.init.xavier_uniform_(self.fc4.weight)

        self.layer4=torch.nn.Sequential(
            self.fc4,
            torch.nn.ReLU(),
            torch.nn.Dropout(p=1-self.keep_prob)
        )
        self.fc5=torch.nn.Linear(80,self.out_dim,bias=True)
        torch.nn.init.xavier_uniform_(self.fc5.weight)
```

3. optimizer설정을 통해 정규화시키기

성능을 더 좋게 만들기 위해 learning scheduler와 adam을 이용한 optimizer를 추가하였고,

Epoch을 늘릴수록 성능이 계속 좋아져 5번에서 10번으로 바꾸었습니다.

```
model=CNN()
criterion=torch.nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=learning_rate)
scheduler=torch.optim.lr_scheduler.LambdaLR(optimizer=optimizer, lr_lambda=lambda epoch: 0.95** epoch,
|                                    last_epoch=-1)
```

또한 dropout을 통해 drop_out_probability가 각각 0.3,0.5,0.8 인경우에 따라 train해본후 결과가 다음과 같았고 그중 성능이 제일 좋았던 0.8을 사용하였습니다.

	P=0.3	P=0.5	P=0.8
Accuracy(epoch=5)	0.5369	0.6066	0.655
Accuracy(epoch=10)	0.5704	0.6261	0.702

4. 테스트 해보기