KW_VIP 과제3

2017741009 로봇학부 장우현

*과제 목표

나만의 네트워크를 구성하여, 그것을 통해 주어진 dataset에 있는 벌과 개미자료들을 training하고 validation data에 대한 정확도 평가를 출력해보자.

*구현 방법

```
class ConvNet(nn.Module):
    def __init__(self, num_classes = 2):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3,16,kernel_size=3,padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,stride=2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(16,32,kernel_size=5,padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,stride=2)
        )
        self.fc = nn.Linear(32*56*56,num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0),-1)
        out = self.fc(out)
        return out

#---------------parameters-----------------
num_epochs = 20

batch_size = 100
learning_rate = 0.001
```

CNN클래스이다. 총 두 번의 conv와 maxpooling을 거치는데, 크기가 224인 input 데이터를 첫 번째 layer를 거치면 output의 크기가 112이고 이 output이 두 번째 layer를 거치면서 크기가 56이 된다. 에폭값과 batch, lr값은 위와 같이 초기값 설정을 해주었다.

```
#-------------------------------------------------------------------------
def train_model(model, criterion, optimizer, scheduler, num_epochs):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)
```

```
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    model.load_state_dict(best_model_wts)
    return model
```

dataset 에서 받은 개미와 벌의 데이터들을 에폭값만큼 학습과 검증단계를 거치고 validation data의 정확도를 출력하는 train_model 함수를 구성한다.

```
train_model(model,criterion, optimizer, exp_lr_schelduler, num_epochs)
```

그리고 각각의 매개변수들을 대입하여 train_model 함수를 실행한다.

*과제 결과

```
Epoch 0/19
----------
train Loss: 1.0456 Acc: 0.5246
val Loss: 0.7633 Acc: 0.5490

Epoch 1/19
----------
train Loss: 0.7673 Acc: 0.5861
val Loss: 0.7870 Acc: 0.5229

Epoch 2/19
----------
train Loss: 0.7076 Acc: 0.5697
val Loss: 0.6878 Acc: 0.5686

Epoch 3/19
----------
train Loss: 0.6727 Acc: 0.5984
val Loss: 0.6615 Acc: 0.6078

Epoch 4/19
----------
train Loss: 0.6210 Acc: 0.6598
val Loss: 0.6907 Acc: 0.6013

Epoch 5/19
----------
train Loss: 0.6167 Acc: 0.6352
val Loss: 0.6108 Acc: 0.6928

Epoch 6/19
----------
train Loss: 0.6198 Acc: 0.6393
val Loss: 0.6645 Acc: 0.6732

Epoch 7/19
----------
train Loss: 0.6010 Acc: 0.6475
val Loss: 0.5931 Acc: 0.6732

Epoch 8/19
----------
train Loss: 0.5835 Acc: 0.6967
val Loss: 0.5923 Acc: 0.6797
```

```
Epoch 9/19
----------
train Loss: 0.5598 Acc: 0.6844
val Loss: 0.5852 Acc: 0.6732

Epoch 10/19
----------
train Loss: 0.5613 Acc: 0.6885
val Loss: 0.5803 Acc: 0.6993

Epoch 11/19
----------
train Loss: 0.5431 Acc: 0.6885
val Loss: 0.5907 Acc: 0.6993

Epoch 12/19
----------
train Loss: 0.5400 Acc: 0.7213
val Loss: 0.6031 Acc: 0.6797

Epoch 13/19
----------
train Loss: 0.5459 Acc: 0.7049
val Loss: 0.5899 Acc: 0.6928

Epoch 14/19
----------
train Loss: 0.5354 Acc: 0.7459
val Loss: 0.5899 Acc: 0.6993

Epoch 15/19
----------
train Loss: 0.5310 Acc: 0.6926
val Loss: 0.5899 Acc: 0.6993

Epoch 16/19
----------
train Loss: 0.5413 Acc: 0.6926
val Loss: 0.5904 Acc: 0.6993

Epoch 17/19
----------
train Loss: 0.5437 Acc: 0.7090
val Loss: 0.5903 Acc: 0.7124

Epoch 18/19
----------
train Loss: 0.5434 Acc: 0.6967
val Loss: 0.5900 Acc: 0.7124

Epoch 19/19
----------
train Loss: 0.5286 Acc: 0.7090
val Loss: 0.5898 Acc: 0.7124

Training complete in 2m 55s
Best val Acc: 0.712418

Process finished with exit code 0
```