딥러닝 기초 기말 Project

학번: 20155326

이름: 신 유 승

서약

아래 보고서는 **본인의 힘만으로 작성**해야 하며, 다른 학생에게 질문과 다른 학생의 코드를 참고하는 행위는 모두 금지합니다

* 수업에서 제공한 코드, 노트북은 모두 재활용 가능하며, 카피로 규정하지 않습니다

* 수업 자료 이외에 참고자료가 있다면, 출처와 사용 부분에 모두 표시하는 경우는 모두 합당한 자료로 인정하겠습니다

* 위에 대해서 모두 이해하고 동의했다면, 아래 `서약글`에 다음을 작성해주세요:

"본인은 위 서약글을 이해하고 동의하며, 프로젝트를 수행하는데 있어서 반칙을 할 경우 (제공자 포함) 본 프로젝트에 대한 점수가 반영되지 않는다는 것에 동의합니다."

학번: 20155326

이름: 신 유 승

서약글: 본인은 위 서약글을 이해하고 동의하며, 프로젝트를 수행하는데 있어서 부정행위를 할 경우(제공자 포함)본 프로젝트에 대한 점수가 반영되지 않는다는 것에 동의합니다.

*모든 코드에는 주석을 작성해 주세요

최종 제출시, 본 보고서와 .ipynb 노트북파일, test에 사용한 모델(.pt)파일을 압축해 제출해 주세요.

중요: 사용한 기법은 자신이 이해한 것 만을 사용하세요. 설명하지 않은 기법을 사용하면 그 부분을 제외하고 채점하겠습니다. 예를 들어서 자신의 힘으로 찾은 코드를 이용하려하는 경우 내용을 이해하고 보고서에 이해한 내용이 충분히 설명이 되어야만 사용을 허용합니다.

Step 1: Dataset 준비하기

Step 1: Dataset 준비하기

[3]:
```
# 이미지 리사이즈 및 정규화
transform_train = transforms.Compose([
    #transforms.Resize((128, 128)),
    transforms.RandomAffine(15),
    transforms.ColorJitter(brightness=0.2),
    transforms.RandomPerspective(distortion_scale=0.1, p=0.3),
    transforms.RandomRotation(10),
    #transforms.GaussianBlur(kernel_size=5, sigma=(0.1, 2.0)),
    #transforms.RandomCrop(100),
    transforms.RandomHorizontalFlip(p=0.5),
    #transforms.RandomGrayscale(p=0.2),
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])


transform_val = transforms.Compose([
    transforms.RandomCrop(100),
    transforms.Resize((128,128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])


transform_test = transforms.Compose([
    transforms.Resize((128,128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])


# dataset 준비
train_dir = "./imgs/train"

train_data = torchvision.datasets.ImageFolder(root=train_dir,transform=transform_train)
valid_data = torchvision.datasets.ImageFolder(root=train_dir,transform=transform_val)



class_names = train_data.classes

num_train = len(train_data)
indices = list(range(num_train))
split = int(np.floor(0.2*num_train))

np.random.shuffle(indices)

# dataset_sizes = {}
# dataset_sizes["train"] = int(0.8 * len(train_data))
# dataset_sizes["val"] = len(train_data) - dataset_sizes["train"]
```

Step 2: Dataset 에 대한 Data Loaders 구성

Step 2: Dataset 에 대한 Data Loaders 구성

[4]:
```
train_idx, valid_idx = indices[split:], indices[:split]

train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

dataloaders = {}
dataloaders["train"] = torch.utils.data.DataLoader(train_data,
                                                   batch_size=32,
                                                   sampler=train_sampler,
                                                   num_workers=4,
                                                  )
dataloaders["val"] = torch.utils.data.DataLoader(valid_data,
                                                   batch_size=16,
                                                   sampler=valid_sampler,
                                                   num_workers=4)

# for x in ["train", "val"]:
#     print("Loaded {} images under {}".format(split[x], x))
print("Loaded {} train images".format(len(dataloaders["train"])*32))
print("Loaded {} valid images".format(len(dataloaders["val"])*16))

print("Classes: ")
print(class_names)
```

```
Loaded 1984 train images
Loaded 496 valid images
Classes:
['AMERICAN_PIPIT', 'BALTIMORE_ORIOLE', 'BELTED_KINGFISHER', 'BROWN_THRASHER', 'CALIFORNIA_GULL', '
OODPECKER', 'RUBY_THROATED_HUMMINGBIRD', 'WHITE_NECKED_RAVEN']
```

Visualization (시각화)

[6]:
```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([.5, .5, .5])
    std = np.array([.5, .5, .5])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)

def show_databatch(inputs, classes):
    out = torchvision.utils.make_grid(inputs)
    imshow(out, title=[class_names[x] for x in classes])

# 학습 데이터의 배치
inputs, classes = next(iter(dataloaders['val']))

# 배치로부터 격자 형태의 이미지
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```

- 이미지 출력 테스트(Sample)

데이터 전처리에 대한 설명

위 작성한 코드에 대해서 설명하세요

- Dataset를 분석하는 과정에서 학습데이터셋의 수가 적다는 판단, 데이터 증강을 위해 이미지 transform을 train과 validation에 각각 적용하였습니다. Transform_train에서는 랜덤으로 15도 기울이기, 0.2 밝기로 색상 조절, 0.1왜곡 스케일과 0.3확률로 랜덤 원근법, 10도 랜덤 회전, 0.5 확률로 랜덤 수평 뒤집기, 128x128 이미지 리사이즈, torch 텐서로 전환, 데이터 정규화 등의 과정을 통해 train dataset transform을 적용하였습니다.
  Transform_val에서는 100 사이즈 랜덤으로 자르기, 128x128 이미지 리사이즈, torch 텐서로 전환, 데이터 정규화 과정을 통해 validation dataset transform을 적용하였습니다. Test transform은 평가를 위해 어떠한 이미지 변형을 주지 않고 진행하였습니다. 저장된 이미지 경로를 통해 pytorch에서 제공하는 ImageFolder를 사용하여 transform이 적용된 학습데이터와 검증데이터를 불러왔습니다. 학습데이터와 검증데이터의 비율은 8:2로 전체 데이터의 80%는 학습데이터, 나머지 20%는 검증데이터에 사용하도록 분할하여 데이터셋을 구성하였습니다. epoch마다 데이터를 섞기 위한 train_sampler, valid_sampler를 통해 배치사이즈 32, 프로세스 num_worker=4로 train data를 불러왔고, 배치사이즈 16, 프로세스 num_worker=4로 validation data를 불러왔습니다. 이미지 데이터가 제대로 불러와졌는지 확인하기 위해 불러온 이미지 개수를 출력해보았고, 이미지 데이터에 따른 라벨명도 출력 확인하였습니다. Transforms 적용까지 완벽하게 데이터가 불러와졌는지 시각적 이미지를 보기 위해 pyplot을 사용하여 다시 한번 확인하였습니다.

Step 3: Neural Network 생성

- Pretrained model을 허용하지 않습니다. (직접 모델을 설계해 주세요)

[4]:
```
class USNet(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv = nn.Sequential(
            # 3x128x128 input image 입력
            # kernel_size 3x3 인 32개의 특징맵(feature map)으로 Conv필터 연산, padding은 1로하여 연산 진행
            nn.Conv2d(3,32,3, padding=1),
            # 연산된 평균과 분산을 조정하기 위한 BatchNormalization
            nn.BatchNorm2d(32),
            # 연산 값이 0보다 크면 그대로 값, 0보다 작으면 0 값을 반환하는 활성 함수 ReLU 적용
            nn.ReLU(True),
            # 정해진 filter 크기 안에서 가장 큰 값만 뽑아냄
            nn.MaxPool2d(2, stride=2),

            # 첫번째 레이어 연산 후 크기
            # 32x64x64 입력
            # kernel_size 3x3 인 64개의 특징맵(feature map)으로 Conv필터 연산, padding은 1로하여 연산 진행
            nn.Conv2d(32,64,3, padding=1),
            # 연산된 평균과 분산을 조정하기 위한 BatchNormalization
            nn.BatchNorm2d(64),
            # 연산 값이 0보다 크면 그대로 값, 0보다 작으면 0 값을 반환하는 활성 함수 ReLU 적용
            nn.ReLU(True),
            # 정해진 filter 크기 안에서 가장 큰 값만 뽑아냄
            nn.MaxPool2d(2, stride=2),
            # 과적합을 방지하기 위해 0.25 만큼 뉴런들을 누락시키기 위한 드롭아웃(Dropout) 적용
            nn.Dropout(0.25),

            # 두번째 레이어 연산 후 크기
            # 64x32x32 입력
            # kernel_size 3x3 인 128개의 특징맵(feature map)으로 Conv필터 연산, padding은 1로하여 연산 진행
            nn.Conv2d(64,128,3, padding=1),
            # 연산된 평균과 분산을 조정하기 위한 BatchNormalization
            nn.BatchNorm2d(128),
            # 연산 값이 0보다 크면 그대로 값, 0보다 작으면 0 값을 반환하는 활성 함수 ReLU 적용
            nn.ReLU(True),
            # 정해진 filter 크기 안에서 가장 큰 값만 뽑아냄
            nn.MaxPool2d(2, stride=2),

            # 세번째 레이어 연산 후 크기
            # 128*16*16 입력
            # kernel_size 3x3 인 256개의 특징맵(feature map)으로 Conv필터 연산, padding은 1로하여 연산 진행
            nn.Conv2d(128,256,3, padding=1),
            # 연산된 평균과 분산을 조정하기 위한 BatchNormalization
            nn.BatchNorm2d(256),
            # 연산 값이 0보다 크면 그대로 값, 0보다 작으면 0 값을 반환하는 활성 함수 ReLU 적용
            nn.ReLU(True),
            # 정해진 filter 크기 안에서 가장 큰 값만 뽑아냄
            nn.MaxPool2d(2, stride=2),
            # 과적합을 방지하기 위해 0.25 만큼 뉴런들을 누락시키기 위한 드롭아웃(Dropout) 적용
            nn.Dropout(0.25),

            # 네번째 레이어 연산 후 크기
            # 256*8*8 입력
            # kernel_size 3x3 인 512개의 특징맵(feature map)으로 Conv필터 연산, padding은 1로하여 연산 진행
            nn.Conv2d(256,512,3, padding=1),
            # 연산된 평균과 분산을 조정하기 위한 BatchNormalization
            nn.BatchNorm2d(512),
            # 연산 값이 0보다 크면 그대로 값, 0보다 작으면 0 값을 반환하는 활성 함수 ReLU 적용
            nn.ReLU(True),
            # 정해진 filter 크기 안에서 가장 큰 값만 뽑아냄
            nn.MaxPool2d(2, stride=2),


            # Conv 연산 후 최종 Output
            # 512*4*4
        )

        # 컨볼루션 연산을 거친 이미지를 다음 일반 신경망을 거치도록 Linear 정의
        # 앞 계층의 출력 크기 512x4x4를 입력, 출력 크기를 4096으로 설정
        self.fc1 = nn.Linear(512*4*4, 4096)
        # 두번째 Linear 또한 앞 크기 4096를 입력, 출력 크기를 15(새 종류 15 클래스)로 설정
        self.fc2 = nn.Linear(4096, 15)

        #activation function (ReLU)
        self.relu = nn.ReLU(True)

        #dropout layer
        self.dropout = nn.Dropout(0.5)
        #Batch Normalization
        self.BN = nn.BatchNorm1d(4096)
```

```
    def forward(self, x):
        # 위 컨볼루션 연산을 통해 얻어진 특징 맵(Feature map) x
        x = self.conv(x)

        # flatten image layer
        # 2차원 특징 맵을 view()함수를 통해 1차원으로 flatten
        # x.size(0)은 x가 가진 원소 개수, -1은 남는 차원 전부를 뜻함
        x = x.view(x.size(0), -1)
        # 1차원으로 펴준 x를 다시 첫번째 Linear에 입력, Batch Normalization와 ReLU를 거침
        x = self.relu(self.BN(self.fc1(x)))
        # 과적합 방지를 위해 0.5만큼 뉴런들을 누락시키기 위한 드롭아웃(Dropout) 적용
        x = self.dropout(x)
        # 마지막 Linear 계층을 통해 출력값 반환
        x = self.fc2(x)

        return x

# 완성된 모델 USNet 생성(create a complete my model)
model = USNet()

# Cuda로 장치 이동(move tensors to GPU if CUDA is available.)
model.to(device)
```

[86]:
```
USNet(
  (conv): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Dropout(p=0.25, inplace=False)
    (9): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (13): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Dropout(p=0.25, inplace=False)
    (18): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (20): ReLU(inplace=True)
    (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=8192, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=15, bias=True)
  (relu): ReLU(inplace=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (BN): BatchNorm1d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

설계한 모델을 출력 후 네트워크를 구성한 방법과 이유를 각 단계별로 설명

Ex) 차원 분석, 채널 수, 커널 크기, linear layer neuron 수

activation function과 선정 이유

최종 layer에서 사용한 activation function이 무엇인지 왜 사용하였는지

CNN 모델이 무엇을 입력 받고 출력하나요? 등...

<서술형>

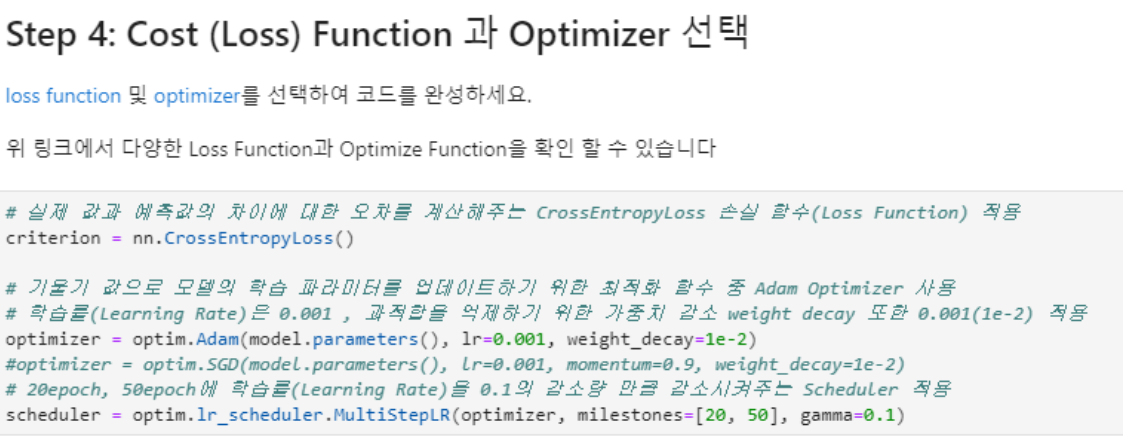
학습하고자 하는 이미지 데이터의 크기는 3x128x128 입니다. 3x128x128의 이미지 데이터를 입력받아 입력 데이터를 컨볼루션 연산하기 위해 첫번째 레이어 블록 단계에서 kernel size 3x3인 32개의 특징맵(feature map)으로 Conv 필터 연산을 진행하였습니다. Convolution 연산을 거친 값의 평균과 분산을 조정하기 위해 배치 정규화(Batch Normalization)을 진행하였습니다. 연산된 값이 0보다 크면 그대로의 값, 0보다 작으면 0 값을 반환하는 활성 함수(Activation Function) ReLU를 적용하였습니다. 다음으로 정해진 필터 크기 안에서 가장 큰 값만 뽑아내기 위해 Maxpooling으로 특징맵을 추출하였습니다. 첫번째 레이어를 통해 연산된 크기는 32x64x64가 되고 다시 두번째 레이어에 입력하여 64개의 특징맵을 갖는 Conv 필터 연산, 배치 정규화, ReLU 활성 함수, Max-pooling을 통해 연산 과정을 진행하였습니다. 두번째 레이어에서는 첫번째 레이어와 다르게 마지막 연산 과정에서 과적합을 방지하기 위해 0.25 만큼 뉴런들을 누락시키기 위한 드롭아웃(Dropout)을 적용하였습니다. 두번째 레이어 또한 연산 후 크기는 64x32x32가 되고, 이를 다시 세번째 레이어에 입력하여 128개의 특징맵을 갖는 Conv 필터 연산, 배치 정규화, ReLU 활성 함수, Max-pooling을 통해 128x16x16 크기의 연산 값을 네번째 레이어에 입력, 256개의 특징맵을 갖는 Conv 필터 연산, 배치 정규화, ReLU 활성 함수, Max-pooling, 드롭아웃을 거쳐 256x8x8 크기의 연산 값을 얻게 됩니다. 256x8x8 크기를 입력 크기로 마지막 레이어에서 512개의 특징맵을 갖는 Conv 필터 연산, 배치 정규화, ReLU 활성 함수, Max-pooling을 통해 최종 Output 결과로 512x4x4의 크기를 갖는 이미지를 갖게 됩니다. 이렇게 컨볼루션 계층과 드롭아웃 등을 거친 이미지는 일반 신경망을 거치게 되는데 앞 계층의 출력 크기인 512x4x4를 입력하여 4096의 크기를 출력하도록 첫번째 Linear를 생성하였습니다. 두번째 Linear 또한 앞 크기의 4096을 입력, 새의 15 가지의 종류를 분류하고자 출력 크기를 15로 정의하였습니다.

설계한 모델을 순차적으로 연산을 진행하기 위해 forward 함수를 사용하여 입력 받는 이미지 x를 위 컨볼루션 연산을 통해 얻어진 특징 맵 x으로 다시 정의하고, 2차원 특징 맵 x를 view 함수를 통해 1차원으로 flatten 하였습니다. 1차원으로 펴준 x를 다시 첫번째 Linear에 입력, Batch Normalization와 ReLU를 거치고 과적합 방지를 위해 0.5만큼 뉴런들을 누락시키기 위한 드롭아

웃(Dropout)을 적용 후, 마지막 Linear 계층을 통해 출력 크기 15인 최종 출력 값을 반환하는 최종 모델 USNet을 구현하였습니다. 완성된 모델을 생성하고 Cuda로 장치를 이동하여 학습 단계 전 모델을 준비하였습니다.

*모델 설계 과정에 사용된 활성화 함수는 입력 값을 비선형한 방식으로 출력 값을 도출하기 위해 사용하였습니다. 활성화 함수에는 여러 종류의 활성 함수가 존재합니다. 그러나 ReLU 함수를 사용한 이유는 ReLU 함수는 계산이 매우 효율적이고, 수렴 속도가 Sigmoid 활성 함수 종류에 비해 6배 정도가 빠르다는 장점이 있기 때문에 사용하였습니다.

Step 4: Cost (Loss) Function 과 Optimizer 선택



Step 4: Cost (Loss) Function 과 Optimizer 선택

loss function 및 optimizer를 선택하여 코드를 완성하세요.

위 링크에서 다양한 Loss Function과 Optimize Function을 확인 할 수 있습니다

[5]:
```
# 실제 값과 예측값의 차이에 대한 오차를 계산해주는 CrossEntropyLoss 손실 함수(Loss Function) 적용
criterion = nn.CrossEntropyLoss()

# 기울기 값으로 모델의 학습 파라미터를 업데이트하기 위한 최적화 함수 중 Adam Optimizer 사용
# 학습률(Learning Rate)은 0.001 , 과적합을 억제하기 위한 가중치 감소 weight decay 또한 0.001(1e-2) 적용
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-2)
#optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight_decay=1e-2)
# 20epoch, 50epoch에 학습률(Learning Rate)을 0.1의 감소량 만큼 감소시켜주는 Scheduler 적용
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[20, 50], gamma=0.1)
```

Optimizer와 Cost 함수를 선정한 이유와 선정하는데 중요하다고 생각하는 내용을 모두 작성합니다.

<서술형>

학습 과정에서 실제 값과 예측 값의 차이에 대한 오차를 계산해주기 위해 손실 함수(Loss Function)을 사용합니다. 이미지 분류 딥러닝 모델은 Linear Model을 통해 최종 값 (logit)이 나오고, softmax 함수를 통해 총합이 1이 되는 0 ~ 1 사이의 값으로 출력한 후 정답 라벨과 Cross Entropy를 통해 loss를 구하게 됩니다. 따라서 이미지 다중 분류 모델에 많이 사용되는 CrossEntropyLoss 함수를 선정하여 적용하였습니다. Optimizer는 기울기 값으로 모델의 학습 파라미터를 업데이트하기 위한 최적화 함수입니다. 기울기 값의 반대 방향과 스탭 사이즈도 적절하게

업데이트를 하기 위해 RMSProp + Momentum을 섞은 기법인 Adam Optimizer를 선정하였습니다. 매개변수로 학습률(Learning Rate)은 0.001, 과적합을 억제하기 위한 가중치 감소 weight decay 또한 0.001(1e-2)를 적용하였습니다. 또한 학습 과정에서 20epoch, 50epoch 에 학습률(Learning Rate)을 0.1의 감소량 만큼 감소시켜주는 스케줄러(Scheduler)를 적용하여 학습의 최적화를 위해 사용하였습니다.

Step 5: 구성한 모델에 대한 Train and Validate 진행

Step 5: 구성한 모델에 대한 Train and Validate 진행

- 코드 전체를 주석으로 설명하세요
- Epoch 별로 Loss나 Accuracy를 출력하여 학습 진행 과정을 확인 할 수 있도록 합니다
- 출력 예시는 주어지나 정해진 형식은 없습니다
- 최적의 모델 저장

예제:

```
Started Training...
Epoch: 1     Training Loss: 3.317162     Validation Loss: 4.162958
Epoch: 2     Training Loss: 2.420140     Validation Loss: 4.182362
...
...
Finished training
```

[20]:
```python
# 학습 진행률을 보기 위해 진행상태바 라이브러리 tqdm 사용
from tqdm.notebook import tqdm, trange

# Hyper Parameter tuning
# epoch 횟수
n_epochs = 500

# track change in validation loss
# validation loss, accuracy 값 초기화
valid_loss_min = np.Inf
valid_acc_max = 0

# track change in train loss
# train loss, accuracy 값 초기화
train_loss_min = np.Inf
train_acc_max = 0

# keep track of training and validation loss
# train, valid loss와 accuracy 초기화
train_loss = torch.zeros(n_epochs)
valid_loss = torch.zeros(n_epochs)

train_acc = torch.zeros(n_epochs)
valid_acc = torch.zeros(n_epochs)
```

```python
for i in trange(0, n_epochs):
    # epoch 수 확인을 위한 출력
    print('epoch = ', i)

    # train model
    model.train()
    # train data 학습
    for data, labels in tqdm(dataloaders["train"]):
        # move tensors to GPU if CUDA is available
        data, labels = data.to(device), labels.to(device)

        # clear the gradients of all optimized variables
        # 모든 최적화된 변수들의 기울기 값 초기화
        optimizer.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        # 예측된 결과 값을 연산하기 위한 forward 진행
        logits = model(data)
        # calculate the batch loss
        # loss 값 계산
        loss = criterion(logits, labels)
        # backward pass: compute gradient of the loss with respect to model parameters
        # 모델 매개 변수에 대한 손실 기울기 계산을 위해 역전파 진행
        loss.backward()
        # perform a single optimization step (parameter update)
        # 파라미터 업데이트를 위한 최적화 단계 수행
        optimizer.step()
        # update training loss
        train_loss[i] += loss.item()

        # 예측된 결과 값 (logits)를 softmax 함수를 통해 총합이 1이 되는 0~1 사이의 값으로 ps 저장
        ps = F.softmax(logits, dim=1)
        # 저장된 ps 값 중에 가장 높은 값 top_p, 해당 클래스 top_class에 저장
        top_p, top_class = ps.topk(1, dim=1)
        # 예측된 top_class 라벨과 실제 라벨이 일치하면 equals는 true
        equals = top_class == labels.reshape(top_class.shape)
        # equals 논리값을 실수형으로 변환 후 맞춘 개수 train_acc에 저장
        train_acc[i] += torch.mean(equals.type(torch.float)).detach().cpu()

    # i번째 epoch에 따른 train loss와 train_acc 계산
    train_loss[i] /= len(dataloaders["train"])
    train_acc[i] /= len(dataloaders["train"])
```

```
    # validate the model
    with torch.no_grad():
        # 모델 평가 진행
        model.eval()
        # validation data 평가
        for data, labels in tqdm(dataloaders["val"]):
            # move tensors to GPU if CUDA is available
            data, labels = data.to(device), labels.to(device)

            # forward pass: compute predicted outputs by passing inputs to the model
            # 예측된 결과 값을 연산하기 위한 forward 진행
            logits = model(data)
            # calculate the batch loss
            # loss 값 계산
            loss = criterion(logits, labels)
            # update average validation loss
            # 평균 validation loss 업데이트
            valid_loss[i] += loss.item()

            # 예측된 결과 값 (logits)를 softmax 함수를 통해 총합이 1이 되는 0~1 사이의 값으로 ps 저장
            ps = F.softmax(logits, dim=1)
            # 저장된 ps 값 중에 가장 높은 값 top_p, 해당 클래스 top_class에 저장
            top_p, top_class = ps.topk(1, dim=1)
            # 예측된 top_class 라벨과 실제 라벨이 일치하면 equals는 true
            equals = top_class == labels.reshape(top_class.shape)
            # equals 논리값을 실수형으로 변환 후 맞춘 개수 valid_acc에 저장
            valid_acc[i] += torch.mean(equals.type(torch.float)).detach().cpu()

    # calculate average losses
    # i번째 epoch에 따른 valid loss와 valid_acc 계산
    valid_loss[i] /= len(dataloaders["val"])
    valid_acc[i] /= len(dataloaders["val"])
```

```
    # print training/validation statistics
    # epoch 마다 training loss 와 validation loss 출력
    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(i, train_loss[i], valid_loss[i]))

    # print training/validation statistics
    # epoch 마다 training accuracy 와 validation accuracy 출력
    print('Epoch: {} \tTraining accuracy: {:.6f} \tValidation accuracy: {:.6f}'.format(i, train_acc[i], valid_acc[i]))

    # scheduler 수행
    scheduler.step()

    # save model if validation loss has decreased
    # validation loss 가 감소하면 모델을 저장
    if valid_loss[i] <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(valid_loss_min,valid_loss[i]))
        torch.save(model.state_dict(), 'model_useung.pt')
        valid_loss_min = valid_loss[i]
        train_loss_min = train_loss[i]
        valid_acc_max = valid_acc[i]
        train_acc_max = train_acc[i]


# 학습 완료 후 최종 모델
print("...Training Finish\n\n")
print("================Best Model================\n")

print('Loss Score : \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(train_loss_min, valid_loss_min))

# print training/validation statistics
print('Accuracy Score : \tTraining accuracy: {:.6f} \tValidation accuracy: {:.6f}'.format(train_acc_max, valid_acc_max))
print("\n==========================================")
```

Step 6: CNN model training/validation 분석

<validation accuracy : 0.836694>

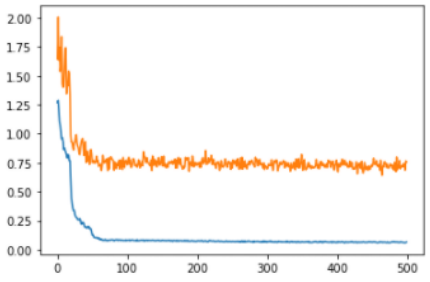
```
100% 62/62 [00:03<00:00, 43.15it/s]
100% 31/31 [00:02<00:00, 13.03it/s]
Epoch: 497      Training Loss: 0.061727         Validation Loss: 0.686699
Epoch: 497      Training accuracy: 1.000000     Validation accuracy: 0.847395
epoch =  498
100% 62/62 [00:03<00:00, 42.84it/s]
100% 31/31 [00:02<00:00, 12.44it/s]
Epoch: 498      Training Loss: 0.061361         Validation Loss: 0.764885
Epoch: 498      Training accuracy: 1.000000     Validation accuracy: 0.808468
epoch =  499
100% 62/62 [00:03<00:00, 43.44it/s]
100% 31/31 [00:02<00:00, 12.49it/s]
Epoch: 499      Training Loss: 0.065492         Validation Loss: 0.756297
Epoch: 499      Training accuracy: 0.999496     Validation accuracy: 0.801489
...Training Finish


================Best Model================

Loss Score :    Training Loss: 0.066105         Validation Loss: 0.642376
Accuracy Score :        Training accuracy: 0.999496     Validation accuracy: 0.836694

==========================================
```

Step 6: CNN model training/validation 분석

- tranining loss와 validation loss 그래프를 통해서 분석

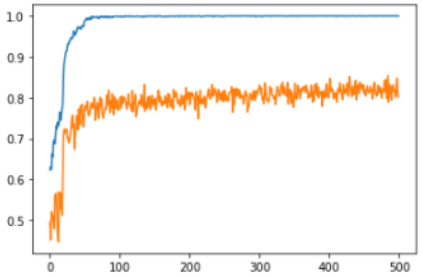
[13]:
```
#코드작성
import matplotlib.pyplot as plt

plt.plot(train_loss)
plt.plot(valid_loss)
```

[13]:
```
[<matplotlib.lines.Line2D at 0x22e06efcdf0>]
```



[14]:
```
plt.plot(train_acc)
plt.plot(valid_acc)
```

[14]:
```
[<matplotlib.lines.Line2D at 0x22e06fbf9a0>]
```



<validation accuracy : 0.860372>

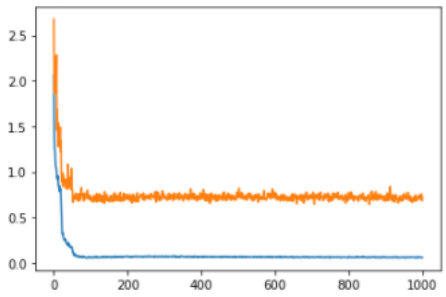
```
100% 54/54 [00:03<00:00, 39.96it/s]

100% 47/47 [00:02<00:00, 27.34it/s]

Epoch: 997      Training Loss: 0.059814         Validation Loss: 0.756452
Epoch: 997      Training accuracy: 1.000000     Validation accuracy: 0.803191
epoch =  998

100% 54/54 [00:03<00:00, 39.74it/s]

100% 47/47 [00:02<00:00, 27.33it/s]

Epoch: 998      Training Loss: 0.061450         Validation Loss: 0.679990
Epoch: 998      Training accuracy: 1.000000     Validation accuracy: 0.837766
epoch =  999

100% 54/54 [00:03<00:00, 40.22it/s]

100% 47/47 [00:02<00:00, 27.37it/s]

Epoch: 999      Training Loss: 0.056486         Validation Loss: 0.720105
Epoch: 999      Training accuracy: 1.000000     Validation accuracy: 0.828457
...Training Finish


================Best Model================

Loss Score :    Training Loss: 0.058678         Validation Loss: 0.643679
Accuracy Score :        Training accuracy: 1.000000     Validation accuracy: 0.860372

==========================================
```

Step 6: CNN model training/validation 분석

- tranining loss와 validation loss 그래프를 통해서 분석

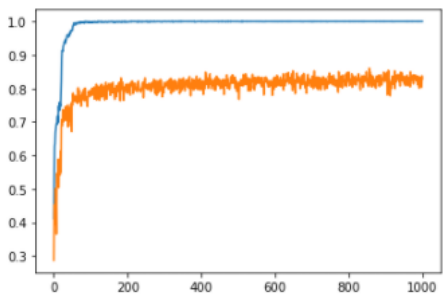
[10]:
```
#코드작성
import matplotlib.pyplot as plt

plt.plot(train_loss)
plt.plot(valid_loss)
```

[10]:
```
[<matplotlib.lines.Line2D at 0x149ff2bb6d0>]
```



[11]:
```
plt.plot(train_acc)
plt.plot(valid_acc)
```

[11]:
```
[<matplotlib.lines.Line2D at 0x149ff527550>]
```



위에서 수행한 training + validation 과정을 설명하세요

training loss와 validation loss 그래프를 통해서 분석

Ex) hyper-parameter, model을 변경하면서 성능 개선한 과정을 최대한 설명하세요

overfitting, underfitting 분석 등..

<서술형>

현재 모델로 학습을 진행하기 전 초기 설계했던 모델은 아래 그림을 보면 알 수 있듯이 데이터 증강도 부족했고, 모델 또한 Conv 연산과 ReLU, Pooling을 통해 학습을 진행하였습니다.

```python
# 이미지 리사이즈 및 정규화
transform_train = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomAffine(10),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

transform_test = transforms.Compose([
    transforms.Resize((128,128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])


# dataset 준비
train_dir = "../input/detect-the-bird/imgs/train"

train_data = torchvision.datasets.ImageFolder(root=train_dir,transform=transform_train)
class_names = train_data.classes

dataset_sizes = {}
dataset_sizes["train"] = int(0.8 * len(train_data))
dataset_sizes["val"] = len(train_data) - dataset_sizes["train"]
```

```python
class USNet(nn.Module):
    def __init__(self):
        super().__init__()
        #3x128x128 image
        self.conv1 = nn.Conv2d(3,32,3, padding=1)
        #32x64x64
        self.conv2 = nn.Conv2d(32,64,3, padding=1)
        #64x32x32
        self.conv3 = nn.Conv2d(64,128,3, padding=1)
        #128x16x16
        self.conv4 = nn.Conv2d(128,256,3, padding=1)
        #256x8x8
        self.conv5 = nn.Conv2d(256,256,3, padding=1)
        #256x4x4
        #max pooling layer
        self.pool = nn.MaxPool2d(2, stride=2)
        self.fc1 = nn.Linear(256*4*4, 1000)
        self.fc2 = nn.Linear(1000, 15)

        #dropout layer
        self.dropout = nn.Dropout(0.4)
        self.relu = nn.ReLU(True)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = self.pool(self.relu(self.conv4(x)))
        x = self.pool(self.relu(self.conv5(x)))

        # flatten image layer
        x = x.view(-1, 256*4*4)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x

# create a complete my model
model = USNet()

# move tensors to GPU if CUDA is available.
model.to(device)
```

```
#코드작성
# Hyper Parameter tuning
n_epochs = 30
batch_size = 32
num_workers = 0
```

```
#코드작성
criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)
```

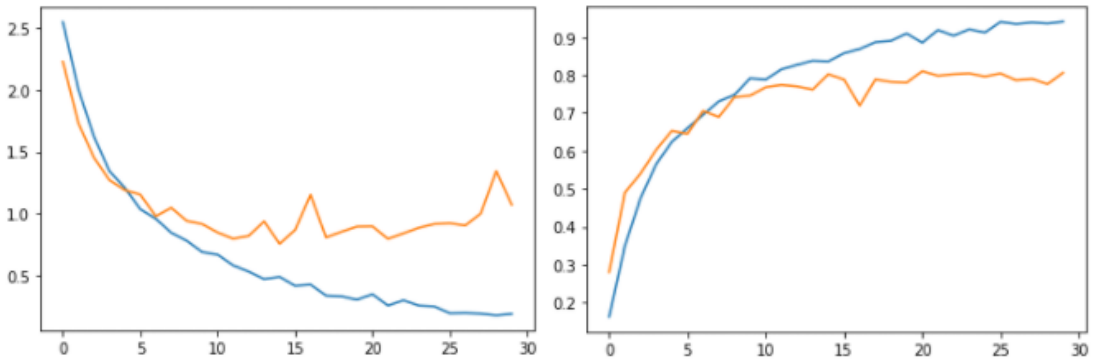
```
...Training Finish
================Best Model================

Loss Score :    Training Loss: 0.491899         Validation Loss: 0.757968
Accuracy Score :        Training accuracy: 0.836370     Validation accuracy: 0.802419
```

<Experiment 1>

위와 같은 hyper parameter 튜닝을 통해 학습을 진행하여

Training accuracy/loss는 83.63%/0.491899 ,

Validation accuracy/loss는 80.24%/0.757968 의 결과를 얻을 수 있었습니다.



그러나 그래프를 통해 학습 분석 결과 training loss와 validation loss의 격차가 벌어지는 것을 보고 오버피팅(Overfitting)이 발생했음을 판단하였고, 역시 제출 결과 Public LeaderBoard 기준 64퍼센트의 성능 결과를 얻게 되었습니다.

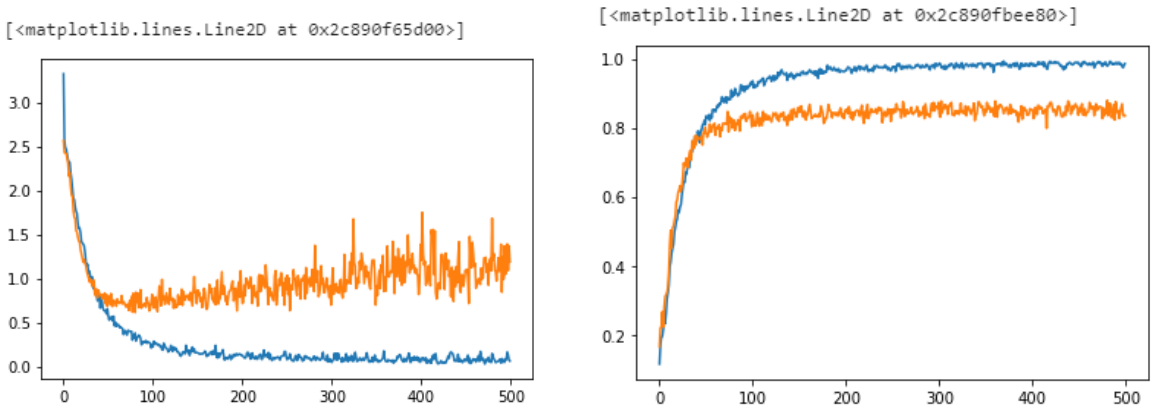
이러한 실패를 기반으로 다양하고 더 많은 데이터 증강을 위해 다양한 transforms 기법을 적용하였고 모델 또한 오버피팅 방지를 위해 Batch Normalization을 추가하여 학습을 진행하였습니다.

```python
# 이미지 리사이즈 및 정규화
transform_train = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomAffine(10),
    transforms.ColorJitter(brightness=0.2),
    #transforms.VerticalFlip(p=0.5),
    transforms.RandomPerspective(distortion_scale=0.1, p=0.3),
    transforms.RandomRotation(10),
    #transforms.GaussianBlur(kernel_size=9, sigma=(0.1, 2.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomGrayscale(p=0.2),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

transform_test = transforms.Compose([
    transforms.Resize((128,128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])


# dataset 준비
train_dir = "./imgs/train"

train_data = torchvision.datasets.ImageFolder(root=train_dir,transform=transform_train)
class_names = train_data.classes

dataset_sizes = {}
dataset_sizes["train"] = int(0.8 * len(train_data))
dataset_sizes["val"] = len(train_data) - dataset_sizes["train"]
```

```python
#코드작성
class USNet(nn.Module):
    def __init__(self):
        super().__init__()
        #3x128x128 image
        self.conv1 = nn.Conv2d(3,32,3, padding=1)
        #32x64x64
        self.conv2 = nn.Conv2d(32,64,3, padding=1)
        #64x32x32
        self.conv3 = nn.Conv2d(64,128,3, padding=1)
        #128x16x16
        self.conv4 = nn.Conv2d(128,256,3, padding=1)
        #256x8x8
        self.conv5 = nn.Conv2d(256,512,3, padding=1)
        #512x4x4
#         self.conv6 = nn.Conv2d(256,512,3, padding=1)
#         #512x2x2

        #max pooling layer
        self.pool = nn.MaxPool2d(2, stride=2)
        self.fc1 = nn.Linear(512*4*4, 4000)
        self.fc2 = nn.Linear(4000, 1000)
        self.fc3 = nn.Linear(1000, 15)

        #dropout layer
        self.dropout = nn.Dropout(0.5)
        self.BN1 = nn.BatchNorm2d(32)
        self.BN2 = nn.BatchNorm2d(64)
        self.BN3 = nn.BatchNorm2d(128)
        self.BN4 = nn.BatchNorm2d(256)
        self.BN5 = nn.BatchNorm2d(512)

        self.relu = nn.ReLU(True)


    def forward(self, x):
        x = self.pool(self.relu(self.BN1(self.conv1(x))))
        x = self.pool(self.relu(self.BN2(self.conv2(x))))
        x = self.pool(self.relu(self.BN3(self.conv3(x))))
        x = self.pool(self.relu(self.BN4(self.conv4(x))))
        x = self.pool(self.relu(self.BN5(self.conv5(x))))


        # flatten image layer
        x = x.view(-1, 512*4*4)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)

        return x

# create a complete my model
model = USNet()

# move tensors to GPU if CUDA is available.
```

```
100%|██████████| 124/124 [00:04<00:00, 27.49it/s]

100%|██████████| 124/124 [00:00<00:00, 135.15it/s]

Epoch: 497      Training Loss: 0.127474         Validation Loss: 1.088512
Epoch: 497      Training accuracy: 0.973958     Validation accuracy: 0.834677
epoch =  498

100%|██████████| 124/124 [00:04<00:00, 28.66it/s]

100%|██████████| 124/124 [00:00<00:00, 132.36it/s]

Epoch: 498      Training Loss: 0.070173         Validation Loss: 1.367212
Epoch: 498      Training accuracy: 0.981351     Validation accuracy: 0.836694
epoch =  499

100%|██████████| 124/124 [00:04<00:00, 27.56it/s]

100%|██████████| 124/124 [00:00<00:00, 136.53it/s]

Epoch: 499      Training Loss: 0.061380         Validation Loss: 1.186265
Epoch: 499      Training accuracy: 0.985383     Validation accuracy: 0.834677
...Training Finish
```

<Experiment 2>



학습 분석 결과 training loss와 accuracy는 수렴하여 잘 학습되었지만, validation loss와 accuracy는 여전히 오버피팅을 해결하지 못한 결과를 도출하였습니다. 제출 결과 Public LeaderBoard 기준 71%의 성능 결과로 이전보다 성능 향상은 있었지만 더 나은 성능 개선을 위해 시도하였습니다.

이러한 문제를 개선하기 위해 Step 3: Neural Network 생성에서 설명한 모델을 통해 학습을 시도하였고, Training accuracy/loss : 99.94%/0.066105 , Validation accuracy/loss : 83.67%/0.642376 결과를 얻었습니다. 이는 epoch 500번에 대한 결과로 학습량을 더 늘리면 개선될 수 있지 않을까 생각하여 동일한 조건으로 epoch 1000번 학습하여 Training accuracy/loss : 100.0%/0.058678 , Validation accuracy/loss : 86.04%/0.643679 결과를 얻었습니다.

두 가지 실험에 따른 최종 제출 결과 Public LeaderBoard 기준 81%와 83%의 성능 결과가 나왔습니다. 이렇게 나온 모델을 가지고 성능 향상을 위해 앙상블 기법을 적용하여 가장 최적의 성능이 나왔던 2개의 모델을 앙상블하여 Predict를 수행하였고, 그 결과 Public LeaderBoard 기준 85.76% 결과가 나오게 되었습니다.

Step 7: Predict with Test Data

- 최적의 모델 로드

[23]:
```
# 학습된 모델 로드
model.load_state_dict(torch.load('model_useung.pt'))
```

[23]:
```
<All keys matched successfully>
```

Step 7: Predict with Test Data

[147]:
```
# test dataset 이미지 불러오기 위한 클래스
class test_dataset(Dataset):
    def __init__(self,imgpath,transform=None):

        self.imgpath = imgpath
        self.transform = transform

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

    def __getitem__(self,idx):
        x = self.transform(Image.open(self.imgpath[idx]).convert('RGB'))

        return x
```

[148]:
```
# test data의 transform 적용
test_transform = transforms.Compose([
        transforms.Resize((128,128)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ])
```

test dataset과 dataloader 생성

[149]:
```
# test 경로를 통해 불러오기
test_set = sorted(glob.glob('./imgs/test/*'))

test_data = test_dataset(test_set,test_transform)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=128, num_workers=0)
```

Predict 수행

[162]:
```
import numpy as np

# 예측값과 최종 예측 결과를 저장하기 위해 빈 배열 생성
pred= []
result = []

# 예측 시작을 알리는 출력문
print("Start Predict...\n\n")
for data in tqdm(test_loader):
    # move tensors to GPU if CUDA is available
    data = data.to(device)
    # forward pass: compute predicted outputs by passing inputs to the model
    # 예측된 결과 값을 연산하기 위한 forward 진행
    logits = model(data)
    # 예측된 결과 값 (logits)를 softmax 함수를 통해 총합이 1이 되는 0~1 사이의 값으로 pred 저장
    pred = logits.softmax(1)
    # 저장된 pred 값 중에 가장 높은 값 prediction 저장
    prediction = torch.argmax(pred, dim=1)
    # 예측 값을 리스트로 변환하여 result에 저장
    result += prediction.detach().cpu().tolist()


# 결과 값 길이 출력 확인
print(len(result))
# 결과 출력
print(result)

# 예측 완료 출력문
print("\n\n...Finish!")
```

```
 40%|████      | 2/5 [00:00<00:00, 10.28it/s]
Start Predict...


100%|██████████| 5/5 [00:00<00:00, 10.78it/s]
604
[9, 8, 9, 13, 13, 11, 14, 3, 14, 6, 13, 7, 1, 8, 11, 14, 6, 9, 2, 1, 7, 0, 0, 11, 0, 7, 1, 6, 0, 2, 7, 0
3, 12, 9, 9, 7, 1, 7, 4, 5, 5, 4, 3, 13, 5, 2, 14, 13, 1, 14, 0, 13, 11, 13, 5, 2, 8, 11, 0, 12, 1, 9, 0
4, 2, 7, 8, 3, 9, 6, 12, 2, 4, 0, 13, 13, 6, 7, 1, 11, 13, 2, 0, 7, 8, 2, 7, 1, 10, 0, 0, 8, 7, 3, 4, 7,
0, 8, 5, 10, 11, 2, 4, 5, 7, 1, 12, 6, 8, 12, 10, 5, 8, 4, 1, 11, 4, 6, 8, 3, 13, 7, 0, 7, 5, 14, 11, 9,
1, 12, 4, 8, 14, 6, 10, 10, 2, 4, 12, 5, 3, 4, 0, 5, 3, 0, 12, 13, 10, 13, 12, 8, 5, 1, 3, 11, 9, 13, 13
9, 9, 14, 5, 7, 6, 4, 12, 1, 6, 9, 2, 1, 13, 4, 0, 11, 11, 7, 1, 7, 10, 3, 8, 11, 3, 9, 10, 9, 10, 11, 6
3, 3, 11, 14, 9, 12, 5, 13, 0, 9, 0, 10, 10, 13, 1, 2, 0, 5, 14, 4, 13, 11, 14, 4, 14, 12, 8, 7, 5, 12,
14, 3, 3, 4, 8, 14, 0, 7, 11, 8, 6, 10, 14, 12, 9, 8, 2, 3, 10, 4, 13, 12, 7, 13, 10, 9, 11, 11, 10, 6,
2, 6, 12, 0, 3, 9, 3, 6, 0, 4, 9, 0, 5, 9, 4, 5, 5, 0, 8, 11, 11, 9, 10, 12, 0, 1, 3, 12, 13, 5, 14, 7,
```

[163]:
```
# class 명 출력 확인
class_names = train_data.classes
print(class_names)
```

```
['AMERICAN_PIPIT', 'BALTIMORE_ORIOLE', 'BELTED_KINGFISHER', 'BROWN_THRASHER', 'CALIFORNIA_GULL', 'CASPIAN_TERN',
OODPECKER', 'RUBY_THROATED_HUMMINGBIRD', 'WHITE_NECKED_RAVEN']
```

[164]:
```
# test 데이터의 이미지 id 와 예측한 class 라벨 값 category를 저장하기 위한 빈 배열 생성
id=[]
category =[]

for i in range(len(test_set)):
    # 불러온 test id 추가
    id.append(test_set[i].split('\\')[-1])
    # result 결과 값을 class names 명으로 추가
    category.append(class_names[result[i]])
# 데이터 프레임을 활용하여 최종 submission csv 파일 저장
pd.DataFrame({'Id':id,'Category':category}).to_csv('useung_sub.csv',index=False)

# 저장 완료 문장 출력
print("Save Complete!")
```

```
Save Complete!
```

[ ]:
```

```

Step 8: Training Techniques

성능 개선을 위해서 사용한 기법 중에서 특별히 효과적이었던 부분이나 강조하고자 하는 내용을 작성해주세요.

Ensemble

[182]:
```
from tqdm import tqdm_notebook
import numpy as np

# 예측값과 최종 예측 결과를 저장하기 위해 빈 배열 생성
result = []
total_pred = np.zeros((604, 15))
# 앙상블하고자 하는 모델 명 저장
name = ['model_useung_81','model_useung_83']
# 예측 값 저장을 위한 빈 배열 생성
pred0 = []
pred1 = []

print("Start Predict...\n\n")
for fold in range(2):
    # USNet 모델 불러옴
    model = USNet()
    # Cuda device로 이동
    model.to(device)
    # name을 통해 앙상블 모델을 하나씩 불러옴
    model.load_state_dict(torch.load('./ensemble/{}.pt'.format(name[fold])))

    # 최종 예측을 위한 빈 배열 생성
    prediction = []
    for idx,data in tqdm_notebook(enumerate(test_loader)):
        # 배치사이즈 만큼 pred 담는 torch 배열 초기화
        pred = torch.zeros((test_loader.batch_size, 15)).to(device)
        # move tensors to GPU if CUDA is available
        data = data.to(device)
        # forward pass: compute predicted outputs by passing inputs to the model
        # 예측된 결과 값을 연산하기 위한 forward 진행
        logits = model(data)
        # 예측된 결과 값 (logits)를 softmax 함수를 통해 총합이 1이 되는 0~1 사이의 값으로 pred 저장
        pred = logits.softmax(1)
        # 예측 값을  numpy로 변환하여 prediction에 추가
        prediction.append(pred.detach().cpu())
```

```
    # 배치 단위로 저장된 prediction을 합침
    prediction = torch.cat(prediction)

    # fold가 0번째일 때, pred0에 prediction 저장
    if fold == 0:
        pred0 = prediction
    # fold가 2번째일 때, pred1에 prediction 저장
    elif fold == 1:
        pred1 = prediction


# 저장된 pred값을 평균내어 가장 높은 값 result에 저장
result = np.argmax((pred0+pred1)/2, axis=1)

# 결과 값 길이 출력 확인
print(len(result))
# 결과 출력
print(result)

# 예측 완료 출력문
print("\n\n...Finish!")
```

```
Start Predict...
```

```
<ipython-input-182-3533ec7f9a2b>:18: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for idx,data in tqdm_notebook(enumerate(test_loader)):
```

```
5/? [00:00<00:00, 10.83it/s]
```

```
5/? [00:00<00:00, 10.42it/s]
```

```
604
tensor([ 9,  8,  9, 13, 13, 11, 14,  3,  5,  6, 13,  7,  1,  8, 11, 14,  6,  9,
         2,  1,  7,  8,  1,  5,  0,  7,  1,  6,  3,  9,  7,  3, 14,  4,  5, 14,
        13, 11,  1,  9,  0,  3, 13,  9,  5, 12,  7,  5, 11,  2,  6,  2, 12,  2,
         5,  4,  7,  3, 12, 12,  6,  1, 13, 13, 14,  5, 11,  7, 12,  3,  3, 12,
         9,  2,  7,  1,  7, 10, 14,  4,  4,  3, 13,  4,  2,  2, 13,  1, 14,  0,
        13, 11, 13,  5,  2, 13, 11,  0, 12,  1,  9,  0, 11,  1, 10, 10, 14, 12,
```

```
...Finish!
```

[183]:
```
# class 명 출력 확인
class_names = train_data.classes
print(class_names)
```

```
['AMERICAN_PIPIT', 'BALTIMORE_ORIOLE', 'BELTED_KINGFISHER', 'BROWN_THRASHER', 'CALIFORNIA_GULL', 'CASH
OODPECKER', 'RUBY_THROATED_HUMMINGBIRD', 'WHITE_NECKED_RAVEN']
```

[184]:
```
# test 데이터의 이미지 id 와 예측한 class 라벨 값 category를 저장하기 위한 빈 배열 생성
id=[]
category =[]
for i in range(len(test_set)):
    # 불러온 test id 추가
    id.append(test_set[i].split('\\')[-1])
    # result 결과 값을 class names 명으로 추가
    category.append(class_names[result[i]])
# 데이터 프레임을 활용하여 최종 submission csv 파일 저장
pd.DataFrame({'Id':id,'Category':category}).to_csv('useung_sub_ensemble.csv',index=False)

# 저장 완료 문장 출력
print("Save Complete!")
```

```
Save Complete!
```

성능 개선을 위해서 기본적으로 hyper parameter 튜닝을 통해 성능 향상을 시도하겠지만, 이를 기반으로 최적의 모델을 통해 앙상블 하는 기법은 기존 성능의 최소 1~3% 향상시킬 수 있는 효

과적인 방법이라고 생각합니다. 앙상블 기법을 통해 성능 개선에 많은 효과를 보았고, 이외에도 Batch Normalization, Scheduler 등을 통해 오버피팅을 방지하고 최적의 성능을 기대할 수 있었습니다.

이러한 여러 가지 성능 개선을 위한 기법들은 대부분 알고 적용하여 다양하게 시도하였기에 특별하지 않을 수 있다고 생각합니다.

그래서 학습한 모델 기반으로 예측한 결과를 Test 데이터와 비교하여 살펴보았고, 예측을 실패하는 이미지를 분석해보았습니다.

분석 결과 학습된 모델이 대체로 이렇게 이미지에서 멀리 위치한 새의 종류를 예측하는 데 어려움을 겪고 있다는 문제점을 파악할 수 있었습니다. 멀리 있는 새의 종류도 예측하기 위해 개선할 수 있는 방법을 고민하다가 학습 후 검증 단계에서 Validation data를 불러올 때, RandomCrop 변형 기법을 적용하여 검증하면 성능의 개선이 있을 것 같다는 생각을 하게 되었고 이를 통해 위의 결과와 같은 성능 개선을 할 수 있었습니다.

Step 1: Dataset 준비하기

[3]:
```
# 이미지 리사이즈 및 정규화
# train data transforms 적용 (랜덤기울이기, 색상 조절, 랜덤원근법, 랜덤회
transform_train = transforms.Compose([
    #transforms.Resize((128, 128)),
    transforms.RandomAffine(15),
    transforms.ColorJitter(brightness=0.2),
    transforms.RandomPerspective(distortion_scale=0.1, p=0.3),
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

# validation data transforms 적용 (100사이즈 기준 랜덤자르기, 이미지 리사
transform_val = transforms.Compose([
    transforms.RandomCrop(100),
    transforms.Resize((128,128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])
```