

영상처리와 딥러닝

20195237 장운성

목차

1. 대회 평가지표 분석
2. Metric learning에 대한 조사 보고서

대회 평가지표 분석

$$MAP@5 = \frac{1}{U} \sum_{u=1}^U \sum_{k=1}^{\min(n,5)} P(k)$$

map는 합성곱 신경망의 모델 성능 평가에 사용된다. map를 이해하기 위해선 precision , recall, ap(average precision)에 대해 알아야 한다.

precision, recall에 대한 블로그 중에 좋은 그림이 있어서 첨부하였습니다.

실제 상황 (ground truth)	예측 결과 (predict result)	
	Positive	Negative
Positive	TP(true Positive) 옳은 검출	FN(false negative) 검출되어야 할 것이 검출되지 않음
Negative	FP(false positive) 틀린 검출	TN(true negative) 검출되지 말아야 할 것이 검출되지 않음

출처 : <https://ctkim.tistory.com/entry/mAPMean-Average-Precision-%EC%A0%95%EB%A6%AC>

이미지를 다루는 합성곱 신경망에선 바운딩 박스를 사용할 때가 많다. 바운딩 박스의 오차를 검증하기 위해 나온 것 중 IOU라는 지표가 있다.

$$IoU = \frac{area(B_{gt} \cap B_p)}{area(B_{gt} \cup B_p)}$$

수식은 위의 그림과 같으며 각각의 바운딩 박스의 교집합을 각각의 바운딩 박스를 합집합 한 값을 나누면 분자 분모 모두 최대 1의 값을 가지며 보통 0.5 이상의 값을 가지면 제대로 검출되었다고 한다. (TP) 0.5 라는 값은 조절 가능하다.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{All\ Detections}$$

precision은 정밀도를 의미한다.

우선 Precision 수식을 보면 정답을 맞춘 경우의 수를 모델이 정답이라고 예측한 경우의 수로 나누어준다.

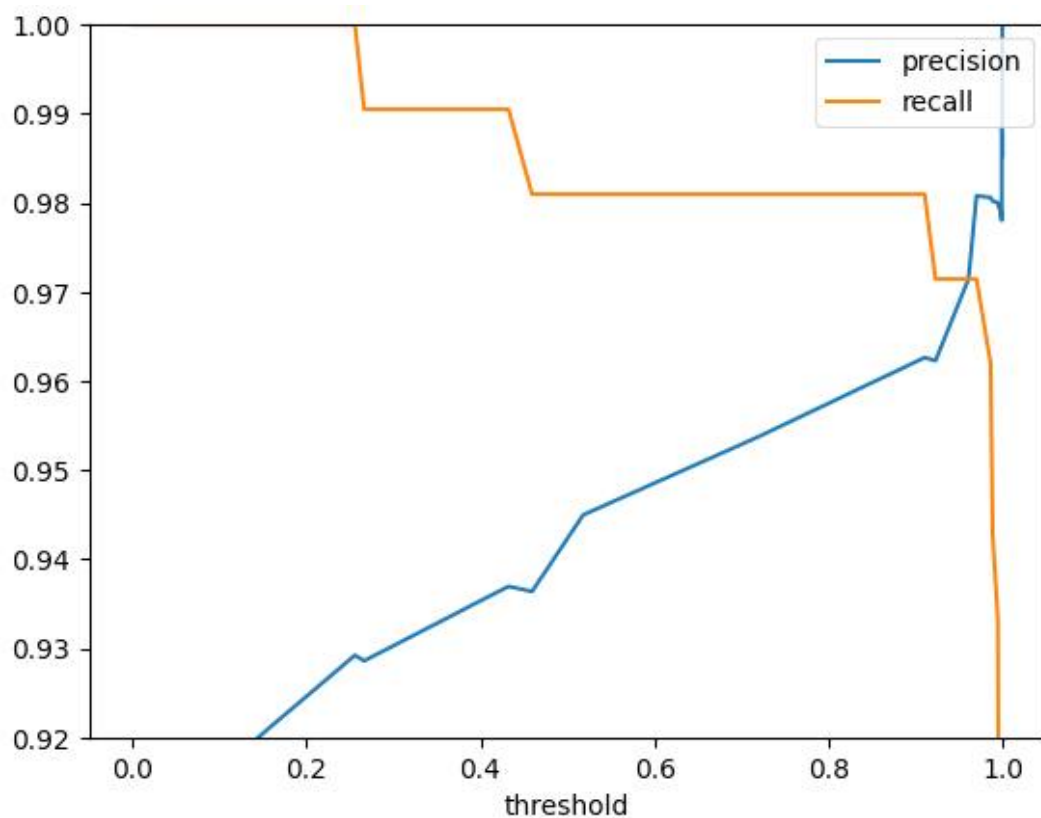
예를 들어 암에 걸린 사람을 5명 검출했는데 그 중 실제 암에 걸린사람이 2명이었다면 precision = 0.4이다. (2/5)

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{All\ Ground\ truths}$$

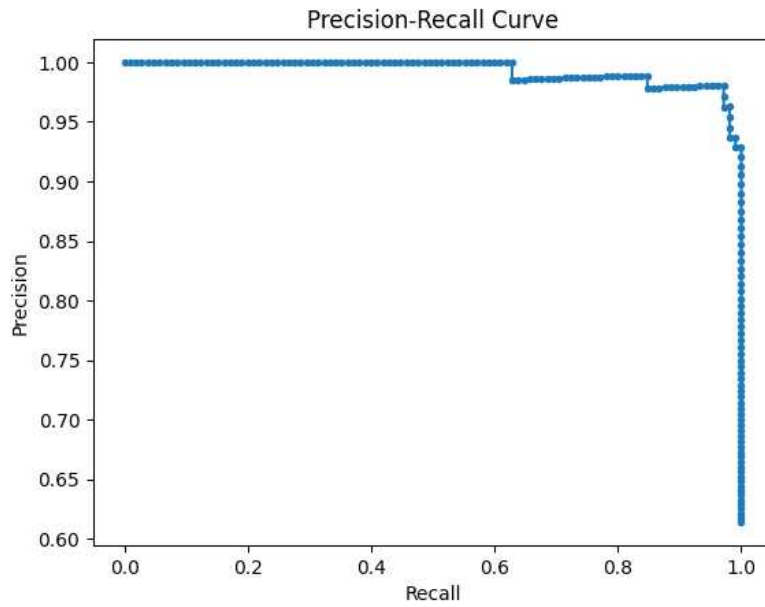
recall은 재현율을 의미한다.

Recall의 수식을 보면 정답을 맞춘 경우의 수를 실제 라벨이 정답인 경우로 나누어준다.

예를들어 5명의 암에 걸린 환자를 검출해야 하는데 2명만 암이라고 검출했다면 recall = 0.4이다. (2/5)



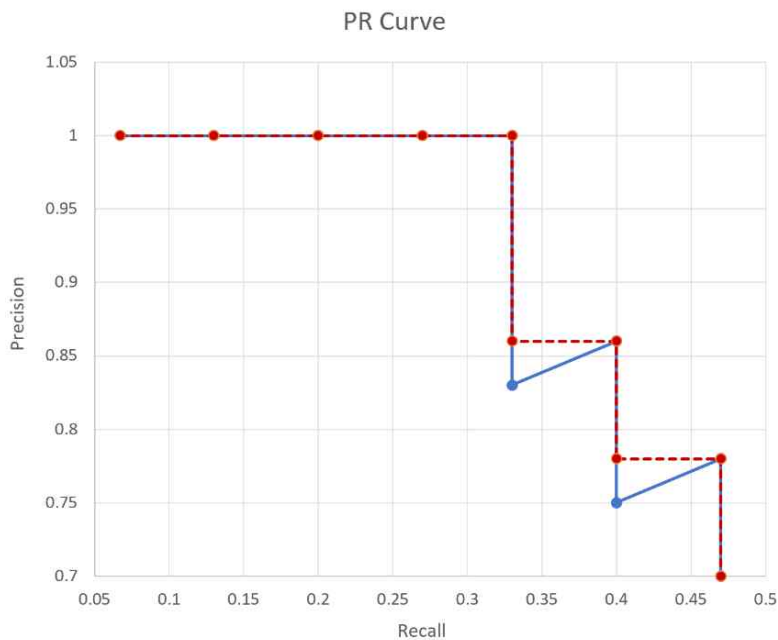
위의 그림은 scikit learn의 위스콘신대 유방암 진단 데이터로 precision과 recall을 나타낸 그래프이다. 일반적으로 그림과 같이 정밀도와 재현율은 서로 반비례 관계를 보인다.



위의 그림은 precision-recall curve를 matplotlib 라이브러리를 사용하여 나타낸 것이다.

해당 그래프를 보면 recall이 커질수록 precision이 감소하는 것을 알 수 있다.

또 중간중간 precision이 뚝 떨어졌다 다시 올라가는 부분이 존재하는데, 이러한 부분들을 고점을 기준으로 잡아 직선으로 연결하여 직사각형을 만든다.



위의 그림을 보면 직사각형 3개가 보일텐데 그 직사각형의 면적을 다 더한 값이 AP이다.

여기에 클래스가 여러개인 경우 각 클래스당 AP를 구한 후 그것들을 모두 합한 뒤 클래스의 개수로 나눠줌으로 알고리즘의 성능을 평가한다.

이것이 바로 mAP(mean average precision)이라고 한다.

Metric Learning

Metric Learning이란?

만약 데이터를 모아 딥러닝의 진행하려 할 때, 매일 분류해야하는 대상이 바뀌는 경우 대표적으로 얼굴인식, 상품 검색 등의 문제를 해결하기 위해 Metric Learning을 한다.

얼굴인식과 상품 검색에는 두가지 공통점이 있다.

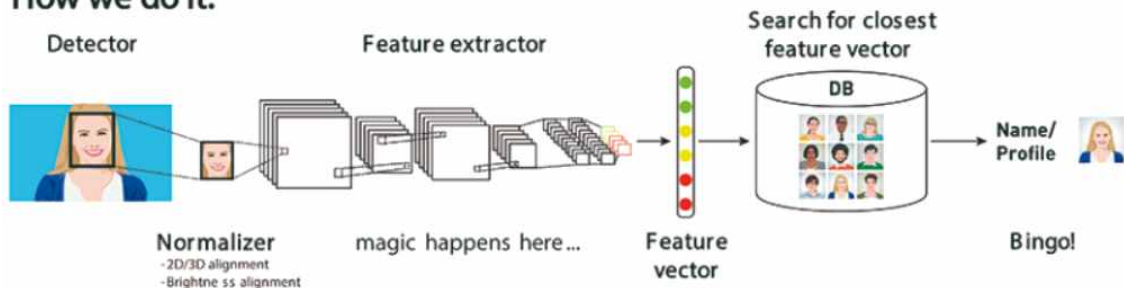
카테고리가 아니라 하나의 인스턴스를 맞추어야 한다는 점과 새로운 인스턴스가 자주 추가되거나 변경이 자주 일어난다는 점이다.

예를 들어, 새로운 출입자가 추가되거나 과거 상품이 제거 될 수 있다.

Metric Learning의 핵심은 크게 두가지다.

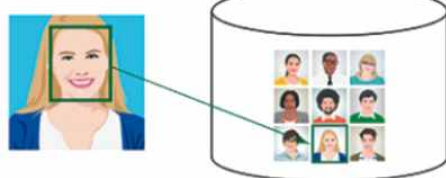
1. 어떤 거리(유사도)를 사용할 것인가? - Euclidean, Cosine Similarity 등
2. 유사도 학습을 위해 어떤 Loss로 학습할 것인가? - Contrastive Loss, Triplet Loss 등

How we do it:

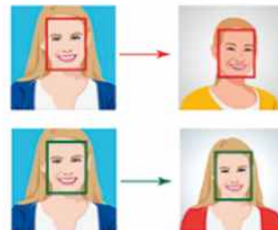


Scenarios:

Identification



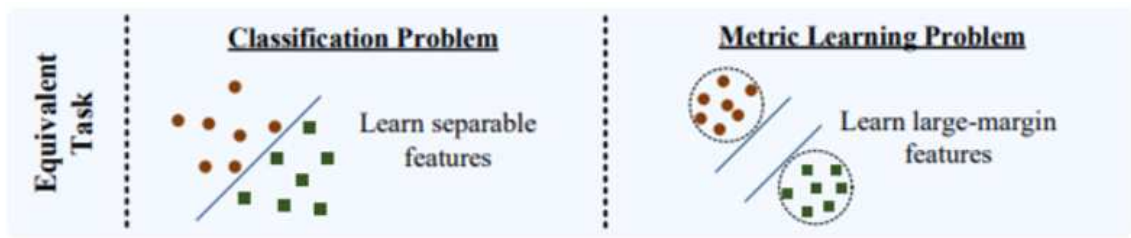
Verification



출처 : <https://kmhana.tistory.com/14>

Metric Learning을 하는법

1. cnn 모델을 활용하여 새로 검색하려는 이미지의 Feature를 추출한다.
2. 추출한 Feature를 사전에 구축된 DB에서 검색한다.
3. 검사 (안면인식 task)
 - Face Identification : 여러 사람 중 'A'라는 사람 찾기
 - Face Verification : 'A'라는 사람이 'A'가 맞는지 확인



Metric Learning의 장점

1. 일반적인 classification은 학습한 이미지에 대해서만 인식이 가능하다.
-> 반면 Metric Learning은 학습하지 않은 이미지도 DB로 구축만 한다면 인식이 가능함
2. 일반적인 classification은 Feature간 Decision Boundary를 찾도록 학습한다.
-> 같은 것에 (정답에) 더 가깝게
반면 Metric Learning은 비 유사한 Feature들을 멀리 떨어지도록 학습한다.
-> 다른 것을 밀어낸다. (Feature 공간을 더 잘 사용할 수 있다.

Humpback Whale Identification Challenge

목차

1. Data Preprocessing
2. Data Augmentation
3. Encoding Labels
4. Dataset, Loaders
5. Initializing Model
6. Model Training (fc-layer)
7. Model Training (all layers)
8. Original Dataset
9. Initializing Main Model
10. Main Model Training (fc-layer)
11. Ensemble

Data Preprocessing

train csv 불러오기

```
train_df = pd.read_csv("C:/Users/COM/Desktop/hlsw/final_project_md/whale-categorization-playground/train.csv")
train_df.head()
```

	Image	Id
0	00022e1a.jpg	w_e15442c
1	000466c4.jpg	w_1287fbc
2	00087b01.jpg	w_da2efe0
3	001296d5.jpg	w_19e5482
4	0014cdf.jpg	w_f22f3e3

train_df의 class 개수 파악

```
There are 9850 images in train dataset with 4251 unique classes.
There are 15610 images in test dataset.
```

train image 확인



train_df의 데이터 분포 확인

```
len(train_df.Id.value_counts())
```

4251

```
for i in range(1, 4):
    print(f'There are {train_df.Id.value_counts()[train_df.Id.value_counts().values==i].shape[0]} classes with {i} samples in train data.')
✓ 0.0s
```

There are 2220 classes with 1 samples in train data.
There are 1034 classes with 2 samples in train data.
There are 492 classes with 3 samples in train data.

총 4251개의 class가 존재하고 첫 번째 class에 대해 2220개의 sample이 존재하며 그 다음부터 1034, 492개의 sample이 존재하는 것을 확인할 수 있습니다.

```
train_df_cnt = train_df.groupby('Id').agg('count').rename({'Image': 'NumImages'}, axis=1)
train_df_cnt.sort_values('NumImages', ascending=False, inplace=True)
train_df_cnt
```

	NumImages
Id	
new_whale	810
w_1287fbc	34
w_98baff9	27
w_7554f44	26
w_1eafe46	23
...	...
w_7e48479	1
w_7e728d8	1
w_7e8305f	1
w_7e841fa	1
w_ffdab7a	1

train_df의 class들을 확인해보니 가장 많은 데이터로 new_whale이 있었고, 그 다음부터 34, 27개 순으로 데이터가 존재하며 해당 고래의 사진이 1장만 존재하는 경우도 많았습니다. 데이터의 분포가 new_whale에 너무 치중되어 있고 고래의 sample이 너무 적은 경우 학습 시에 방해가 될 수 있다고 생각했습니다.

```
train_new_whale = train_df[train_df.Id == "new_whale"].head(18)
```

train data에 new_whale이 많으니 test data에도 어느정도 new_whale이 많이 존재한다고 생각하여 new_whale을 따로 저장하여 나중에 train_df에 concat 해주었습니다.

```
train_df_cnt.drop('new_whale', inplace=True)
train_df_cnt
```

	NumImages
Id	
w_1287fbc	34
w_98baff9	27
w_7554f44	26
w_1eafe46	23
w_fd1cb9d	22
...	...
w_7e48479	1
w_7e728d8	1
w_7e8305f	1
w_7e841fa	1
w_ffdab7a	1

학습에 도움이 되도록 데이터의 불균형을 피하고자 new_whale을 제거하였습니다.

```
NUM_IMAGES_THRESHOLD = 15

over15_df = train_df_cnt.query(f'NumImages > {NUM_IMAGES_THRESHOLD}')

print('shape:', over15_df.shape)
print('total number of images:', over15_df.NumImages.sum())
over15_df
```

NumImages			
Id			
		w_9b401eb	19
		w_c0d494d	18
w_1287fbc	34	w_b7d5069	18
w_98baff9	27	w_0e737d0	17
w_7554f44	26	w_eb0a6ed	17
w_1eafe46	23	w_dbda0d6	17
w_fd1cb9d	22	w_18eee6e	17
w_ab4cae2	22	w_17ee910	16
w_693c9ee	22	w_a59905f	16
w_987a36f	21	w_6c803bf	16
w_43be268	21	w_b0e05b1	16
w_73d5489	21	w_67de30b	16
w_f19faeb	20		
w_95874a5	19		

해당 고래의 이미지가 매우 적은 경우 학습을 진행할 때 특성을 제대로 파악하지 못할 것 같아 15보다 적은 데이터들은 제거하였습니다.

```
len(over15_df)
```

24

15보다 많은 고래의 class들은 24개로 확인하였습니다.

```
ids_to_leave = list(over15_df.index)
train_df = train_df.query(f'Id in {ids_to_leave}')
train_df
```

	Image	Id
1	000466c4.jpg	w_1287fbc
13	00467ae9.jpg	w_fd1cb9d
62	01981e98.jpg	w_f19faeb
81	01ed89f6.jpg	w_67de30b
104	02916b71.jpg	w_73d5489
...
9740	fd30f7e6.jpg	w_9b401eb
9811	fe6c845c.jpg	w_dbda0d6
9814	ff0de88a.jpg	w_18eee6e
9828	ff6946b4.jpg	w_17ee910
9834	ff9d60a3.jpg	w_eb0a6ed

481 rows × 2 columns

15보다 많은 sample을 가진 고래들을 train_df로 바꾸어 주었습니다.

```
train_df.drop(train_df.query("Image == '496b52ff.jpg'").index,
              axis=0, inplace=True)
```



고래의 꼬리보다 배에 비중이 커보이는 해당 사진을 제거하였습니다.

```
train_df = pd.concat([train_df, train_new_whale])
train_df
```

	Image	Id
1	000466c4.jpg	w_1287fbc
13	00467ae9.jpg	w_fd1cb9d
62	01981e98.jpg	w_f19faeb
81	01ed89f6.jpg	w_67de30b
104	02916b71.jpg	w_73d5489
...
148	03c6a5c0.jpg	new_whale
151	03de1a54.jpg	new_whale
184	04a39247.jpg	new_whale
187	04ad14b2.jpg	new_whale
200	04d7f905.jpg	new_whale

498 rows × 2 columns

위에서 처리해두었던 new_whale 데이터 18개를 concat하여 train에서도 new_whale에 대한 정보를 학습할 수 있게 하였습니다.

Data Augmentation

transforms 설정

```
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])
])

data_transforms_test = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])
])
```

augmentation의 경우 RandomResizedCrop, ColorJitter 정도를 추가로 사용해 보았는데, RandomResizedCrop의 경우 모델이 더 세부적인 특징을 뽑아낼 수 있어 성능이 올라갈 거라고 예상했지만, 고래의 전체적인 꼬리 모양과 특정 부분에 고래를 판단하는 핵심 부분이 잘려 학습이 잘 되지 않아 제거하였습니다.

또한, RandomResizedCrop을 사용하였을 때 성능이 좋지 않았기 때문에 이미지의 모양을 크게 변형시키는 augmentation은 사용하지 않았습니다.

ColorJitter는 이미지의 모양을 건들지 않고 augmentation 정도를 파라미터로 정할 수 있어 사용해 보았지만, 성능이 떨어져 제거하였습니다.

test_data에 대해서도 RandomHorizontalFlip을 적용해 주었는데, 이는 데이터의 큰 변형이 아닌 이미지를 뒤집는 간단한 augmentation이라 그대로 두었는데 성능이 나쁘지 않아 그대로 사용했습니다.

Encoding Labels

```
def prepare_labels(y):
    values = np.array(y)
    label_encoder = LabelEncoder()
    integer_encoded = label_encoder.fit_transform(values)

    onehot_encoder = OneHotEncoder(sparse=False)
    integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
    onehot_encoded = onehot_encoder.fit_transform(integer_encoded)

    y = onehot_encoded
    return y, label_encoder
```

```
y, le = prepare_labels(train_df['Id'])
```

label encoding의 경우 파이썬에서 지원하는 LabelEncoder()를 사용하여 y값으로 들어온 train_df의 id값을 카테고리형에서 수치형으로 변환후 OneHotEncoder를 사용하여 one_hot vector로 바꾸어 주었습니다.

Dataset, Loaders

Dataset

```
class WhaleDataset(Dataset):
    def __init__(self, datafolder, datatype='train', df=None, transform = transforms.Compose([transforms.ToTensor()]), y=None):
        self.datafolder = datafolder
        self.datatype = datatype
        self.y = y
        if self.datatype == 'train':
            self.df = df.values
            self.image_files_list = [self.datafolder + s for s in self.df[:,0]]
        else:
            self.image_files_list = [s for s in os.listdir(datafolder)]
        self.transform = transform

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

    def __getitem__(self, idx):
        if self.datatype == 'train':
            img_name = os.path.join(self.datafolder, self.df[idx][0])
            label = self.y[idx]

            elif self.datatype == 'test':
                img_name = os.path.join(self.datafolder, self.image_files_list[idx])
                label = np.zeros((5005,))

            image = Image.open(img_name).convert('RGB')
            image = self.transform(image)
            if self.datatype == 'train':
                return image, label
            elif self.datatype == 'test':
                # so that the images will be in a correct order
                return image, label, self.image_files_list[idx]
```

dataset의 경우에는 datafolder, datatype, y, df, transform을 받아 초기화를 해주었습니다. data type에 따라 train, test set이 달라지는데, train일 경우 dataframe을 같이 받아 초기화 해주었습니다.

image_files_list는 위에서 data preprocessing을 통해 골라낸 데이터만을 가져오기 위해 train_data에 한해서 인덱싱을 통해 같은 이미지만 불러올 수 있게 하였습니다.

getitem 부분에서 os.path.join()을 사용하여 각 폴더의 이미지들의 주소를 가지고 Image.open() 함수를 사용해 image의 형태로 label과 함께 return 해주었습니다.

getitem 또한 train_data에 한해서 해당 dataframe indexing을 통해 파일을 불러왔습니다.

train dataset, test dataset

```
train_dataset = WhaleDataset(datafolder='C:/Users/COM/Desktop/hlsw/final_project_md/whale-categorization-playground/train/',\
                             datatype='train', df=train_df, transform=data_transforms, y=y)
test_set = WhaleDataset(datafolder='C:/Users/COM/Desktop/hlsw/final_project_md/whale-categorization-playground/test/',\
                        datatype='test', transform=data_transforms_test)
```

train, valid dataset split

```
from torch.utils.data import random_split

train_size = 0.75
num_train = int(len(train_df) * train_size)
num_val = len(train_df) - num_train

train_dataset, validation_dataset = random_split(train_dataset, [num_train, num_val])
```

random_split() 함수를 사용하여 train_dataset을 0.75 : 0.25로 데이터를 분할 하였습니다.

Loaders

```
valid_sampler = SubsetRandomSampler(list(range(len(os.listdir('C:/Users/COM/Desktop/hlsu/final_project_md/whale-categorization-playground/test/')))))
batch_size = 64
num_workers = 0

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(validation_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
# less size for test loader.
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32, sampler=valid_sampler, num_workers=num_workers)
```

batch_size는 cuda memory에 맞춰 64로 설정하였습니다.

32를 사용해 보았지만 성능이 더 떨어져서 64로 설정하였습니다.

또한 train, valid loader는 random_split() 함수를 사용하였고, shuffle True를 설정하며 sampler없이 초기화 시켜주었고, test_loader는 subsetRandomSampler를 적용시켜 주었습니다.

Initializing Model

Initializing Small Model

```
model_conv = models.resnet18(pretrained=True) # 모델의 객체 생성

model_conv.fc = nn.Linear(512, 25)

for i, (name, param) in enumerate(model_conv.named_parameters()):
    param.requires_grad = False
    if i == 59:
        print('end')
        break

optimizer = optim.Adam(model_conv.parameters(), lr=0.01)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
criterion = nn.CrossEntropyLoss() # 손실 함수 정의
```

model은 resnet18, resnet50, vgg16을 사용해 보았는데 가장 성능이 좋았던 resnet18 모델을 사용하였습니다.

pretrained model을 사용하였고 fc layer만 train_dataset에 맞게 설정해주었습니다.

해당 모델의 parameter.requires_grad를 false로 해주어 pretrained된 cnn 파라미터들을 고정하고 새로운 데이터셋에 맞게 fc layer만 재학습을 진행하였습니다.

optimizer는 adam을 사용하였고 overfitting 현상이 발생하여 weight_decay 항목도 설정을 해보았으나 학습에 도움이 되지 않아 제거하였습니다.

scheduler는 일정 step마다 lr을 gamma만큼 조정해주는 StepLR을 사용하였습니다.

lossFunction은 흔히 사용하는 CrossEntropyLoss를 사용하였습니다.

```
# 전체 모델의 파라미터를 확인하면서 requires_grad 값을 출력
for idx, (name, param) in enumerate(list(model_conv.named_parameters())):
    print(f"{name}: requires_grad={param.requires_grad}")
```

```
conv1.weight: requires_grad=False
bn1.weight: requires_grad=False
bn1.bias: requires_grad=False
layer1.0.conv1.weight: requires_grad=False
layer1.0.bn1.weight: requires_grad=False
layer1.0.bn1.bias: requires_grad=False
layer1.0.conv2.weight: requires_grad=False
layer1.0.bn2.weight: requires_grad=False
layer1.0.bn2.bias: requires_grad=False
layer1.1.conv1.weight: requires_grad=False
layer1.1.bn1.weight: requires_grad=False
layer1.1.bn1.bias: requires_grad=False
layer1.1.conv2.weight: requires_grad=False
layer1.1.bn2.weight: requires_grad=False
layer1.1.bn2.bias: requires_grad=False
layer2.0.conv1.weight: requires_grad=False
layer2.0.bn1.weight: requires_grad=False
layer2.0.bn1.bias: requires_grad=False
layer2.0.conv2.weight: requires_grad=False
layer2.0.bn2.weight: requires_grad=False
layer2.0.bn2.bias: requires_grad=False
layer2.0.downsample.0.weight: requires_grad=False
layer2.0.downsample.1.weight: requires_grad=False
layer2.0.downsample.1.bias: requires_grad=False
layer2.1.conv1.weight: requires_grad=False
...
layer4.1.bn2.weight: requires_grad=False
layer4.1.bn2.bias: requires_grad=False
fc.weight: requires_grad=True
```

위의 사진과 같이 모델의 파라미터들의 requires_grad가 False로 된 것을 확인할 수 있습니다. (train 시에 학습되지 않음)

Model Training (fc-layer)

```
model_conv=model_conv.cuda()
n_epochs = 20

train_loss = torch.zeros(n_epochs)
valid_loss = torch.zeros(n_epochs)

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

valid_loss_min = np.Inf

for epoch in range(0, n_epochs):
    model_conv.train()
    exp_lr_scheduler.step()

    for batch_i, (data, target) in enumerate(tqdm.tqdm(train_loader)):
        data, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        output = model_conv(data)
        loss = criterion(output, target.float())
        loss.backward()
        optimizer.step()
        train_loss[epoch] += loss.item()

        # ps의 shape: (배치 크기, 클래스 개수)
        ps = F.softmax(output, dim=1)

        # top_p의 shape: (배치 크기, 1), top_class의 shape: (배치 크기, 1)
        top_p, top_class = ps.topk(1, dim=1)
        top_class = top_class.cpu()
        equals = top_class == np.argmax(target.cpu(), axis=1).reshape(top_class.shape)
        # train_acc의 shape: scalar (텐서이지만 스칼라 값)
        train_acc[epoch] += torch.mean(equals.type(torch.float)).detach().cpu()
    train_loss[epoch] /= len(train_loader)
    train_acc[epoch] /= len(train_loader)

    with torch.no_grad():
        model_conv.eval()
        for batch_i, (data, target) in enumerate(tqdm.tqdm(valid_loader)):
            data, target = data.cuda(), target.cuda()

            output = model_conv(data)
            loss = criterion(output, target.float())
            valid_loss[epoch] += loss.item()

            ps = F.softmax(output, dim=1)
            top_p, top_class = ps.topk(1, dim=1)
            top_class = top_class.cpu()
            equals = top_class == np.argmax(target.cpu(), axis=1).reshape(top_class.shape)
            valid_acc[epoch] += torch.mean(equals.type(torch.float)).detach().cpu()

        # calculate average losses
        valid_loss[epoch] /= len(valid_loader)
        valid_acc[epoch] /= len(valid_loader)

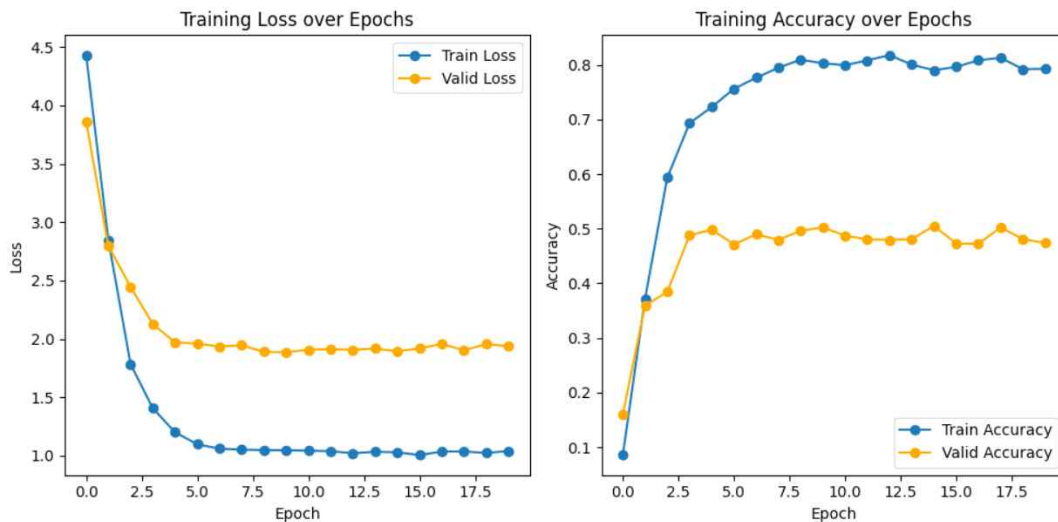
    # exp_lr_scheduler.step()

    print(f'Epoch {epoch+1}, train loss: {train_loss[epoch]:.4f}, train acc: {train_acc[epoch]:.4f}')
    print(f'Epoch {epoch+1}, valid loss: {valid_loss[epoch]:.4f}, valid acc: {valid_acc[epoch]:.4f}')

    if valid_loss[epoch] <= valid_loss_min: # valid_loss가 더 작을경우 모델 저장
        print('Validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
            valid_loss_min,
            valid_loss[epoch]))
        torch.save(model_conv, 'small_model.pt')
        torch.save(model_conv.state_dict(), 'small_model_state.pt')
        valid_loss_min = valid_loss[epoch]
```

train, valid loss & accuracy

```
Epoch 10, train loss: 1.0455, train acc: 0.8029  
Epoch 10, valid loss: 1.8851, valid acc: 0.5022  
Validation loss decreased (1.888769 --> 1.885079). Saving model ...
```



model.load

```
model_conv.load_state_dict(torch.load('small_model_state.pt')) # 가장 좋았던 모델 값 불러오기.
```

training all_layers

```
# 전체 모델의 파라미터를 확인하면서 requires_grad 값을 출력  
for name, param in model_conv.named_parameters():  
    print(f"{name}: requires_grad={param.requires_grad}")  
  
for i, (name, param) in enumerate(model_conv.named_parameters()):  
    param.requires_grad = True  
    if i == 158:  
        print('end')  
        break
```

```
conv1.weight: requires_grad=True  
bn1.weight: requires_grad=True  
bn1.bias: requires_grad=True  
layer1.0.conv1.weight: requires_grad=True  
layer1.0.bn1.weight: requires_grad=True  
layer1.0.bn1.bias: requires_grad=True  
layer1.0.conv2.weight: requires_grad=True  
layer1.0.bn2.weight: requires_grad=True  
layer1.0.bn2.bias: requires_grad=True  
layer1.1.conv1.weight: requires_grad=True  
layer1.1.bn1.weight: requires_grad=True  
layer1.1.bn1.bias: requires_grad=True  
layer4.1.bn2.weight: requires_grad=True  
layer4.1.bn2.bias: requires_grad=True  
fc.weight: requires_grad=True  
fc.bias: requires_grad=True
```


Model Training (all layers)

```
Epoch 2, train loss: 1.0221, train acc: 0.7983
Epoch 2, valid loss: 1.8896, valid acc: 0.4807
Validation loss decreased (1.932840 --> 1.889611). Saving model ...
```

conv layer는 pretrained된 데이터에 맞추어져 있기 때문에 test dataset에 대해 분류를 진행하기 위해 train dataset에 맞게 conv layer 또한 학습을 진행하였습니다.

Original Dataset

model.load

```
model_conv.load_state_dict(torch.load('small_model_state.pt')) # 가장 좋았던 모델 값 불러오기.
```

train csv 불러오기

```
train_df = pd.read_csv("C:/Users/COM/Desktop/hlsw/final_project_md/whale-categorization-playground/train.csv")
train_df.head()
```

	Image	Id
0	00022e1a.jpg	w_e15442c
1	000466c4.jpg	w_1287fbc
2	00087b01.jpg	w_da2efe0
3	001296d5.jpg	w_19e5482
4	0014cfd5.jpg	w_f22f3e3

ship data 제거

```
train_df.drop(train_df.query("Image == '496b52ff.jpg'").index,  
              axis=0, inplace=True)
```

transforms 설정

```
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])

data_transforms_test = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])
```

Encoding Labels

```
def prepare_labels(y):  
    # From here: https://www.kaggle.com/pestipeti/keras-cnn-starter  
    values = np.array(y)  
    label_encoder = LabelEncoder()  
    integer_encoded = label_encoder.fit_transform(values)  
  
    onehot_encoder = OneHotEncoder(sparse=False)  
    integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)  
    onehot_encoded = onehot_encoder.fit_transform(integer_encoded)  
  
    y = onehot_encoded  
    return y, label_encoder  
  
y, le = prepare_labels(train_df['Id'])
```

Dataset 정의

```
class WhaleDataset(Dataset):  
    def __init__(self, datafolder, datatype='train', df=None, transform = transforms.Compose([transforms.ToTensor()]), y=None):  
        self.datafolder = datafolder  
        self.datatype = datatype  
        self.y = y  
        if self.datatype == 'train':  
            self.df = df.values  
            self.image_files_list = [self.datafolder + s for s in self.df[:,0]]  
        else:  
            self.image_files_list = [s for s in os.listdir(datafolder)]  
        self.transform = transform  
  
    def __len__(self):  
        return len(self.image_files_list)  
  
    def __getitem__(self, idx):  
        if self.datatype == 'train':  
            img_name = os.path.join(self.datafolder, self.df[idx][0])  
            label = self.y[idx]  
  
            elif self.datatype == 'test':  
                img_name = os.path.join(self.datafolder, self.image_files_list[idx])  
                label = np.zeros((5005,))  
  
            image = Image.open(img_name).convert('RGB')  
            image = self.transform(image)  
            if self.datatype == 'train':  
                return image, label  
            elif self.datatype == 'test':  
                # so that the images will be in a correct order  
                return image, label, self.image_files_list[idx]
```

train dataset, test dataset

```
train_dataset = WhaleDataset(datafolder='C:/Users/COM/Desktop/hlsf/final_project_md/whale-categorization-playground/train/',  
                             datatype='train', df=train_df, transform=data_transforms, y=y)  
test_set = WhaleDataset(datafolder='C:/Users/COM/Desktop/hlsf/final_project_md/whale-categorization-playground/test/',  
                        datatype='test', transform=data_transforms_test)
```

train, valid dataset split

```
from torch.utils.data import random_split

train_size = 0.8
num_train = int(len(train_df) * train_size)
num_val = len(train_df) - num_train

train_dataset, validation_dataset = random_split(train_dataset, [num_train, num_val])
```

Loaders

```
valid_sampler = SubsetRandomSampler(list(range(len(os.listdir('C:/Users/COM/Desktop/hlsw/final_project_md/whale-categorization-playground/test/')))))
batch_size = 64
num_workers = 0

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(validation_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
# less size for test loader.
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32, sampler=valid_sampler, num_workers=num_workers)
```

model fc layer requires_grad True

```
for i, (name, param) in enumerate(model_conv.named_parameters()):
    param.requires_grad = False
    if i == 59:
        print('end')
        break
```

Enitializing Main Model

```
model_conv.fc = nn.Linear(512, 4251)

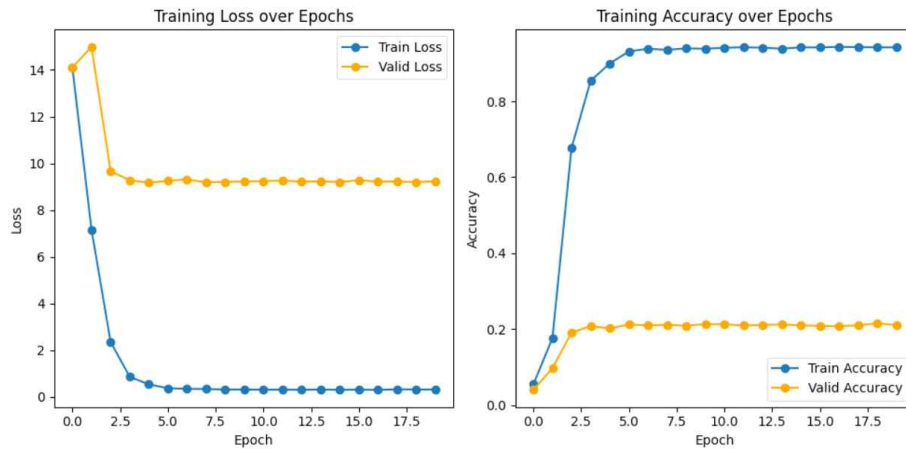
optimizer = optim.Adam(model_conv.fc.parameters(), lr=0.01)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
criterion = nn.CrossEntropyLoss() # 손실 함수 정의
```

전체 dataset에 대한 class가 4251개이므로 fc layer의 output을 수정했습니다.

Main Model Training (fc-layer)

```
Epoch 5, train loss: 0.5336, train acc: 0.8995
Epoch 5, valid loss: 9.1632, valid acc: 0.2021
Validation loss decreased (9.279389 --> 9.163165). Saving model ...
```

train, valid loss & accuracy



Ensemble

```
model1 = models.resnet18(pretrained=True) # 모델의 객체 생성
model1.fc = torch.nn.Linear(512,4251) # 전이학습
model1.load_state_dict(torch.load('resnet18_big_model_0.408_state.pt', map_location='cpu')) #
model2 = models.resnet18(pretrained=True) # 모델의 객체 생성
model2.fc = torch.nn.Linear(512,4251) # 전이학습
model2.load_state_dict(torch.load('resnet18_big_model_0.4181_state.pt', map_location='cpu')) #
model3 = models.resnet18(pretrained=True) # 모델의 객체 생성
model3.fc = torch.nn.Linear(512,4251) # 전이학습
model3.load_state_dict(torch.load('resnet18_big_model_0.41423_state.pt', map_location='cpu')) #
model4 = models.resnet18(pretrained=True) # 모델의 객체 생성
model4.fc = torch.nn.Linear(512,4251) # 전이학습
model4.load_state_dict(torch.load('resnet18_big_model_0.42135_state.pt', map_location='cpu')) #
model5 = models.resnet18(pretrained=True) # 모델의 객체 생성
model5.fc = torch.nn.Linear(512,4251) # 전이학습
model5.load_state_dict(torch.load('resnet18_big_model_0.42845_state.pt', map_location='cpu')) #
model6 = models.resnet18(pretrained=True) # 모델의 객체 생성
model6.fc = torch.nn.Linear(512,4251) # 전이학습
model6.load_state_dict(torch.load('resnet18_big_model_0.42057_state.pt', map_location='cpu')) #
model7 = models.resnet18(pretrained=True) # 모델의 객체 생성
model7.fc = torch.nn.Linear(512,4251) # 전이학습
model7.load_state_dict(torch.load('resnet18_big_model_0.41763_state.pt', map_location='cpu')) #
model8 = models.resnet18(pretrained=True) # 모델의 객체 생성
model8.fc = torch.nn.Linear(512,4251) # 전이학습
model8.load_state_dict(torch.load('resnet18_big_model_0.42073_state.pt', map_location='cpu')) #
model9 = models.resnet18(pretrained=True) # 모델의 객체 생성
model9.fc = torch.nn.Linear(512,4251) # 전이학습
model9.load_state_dict(torch.load('resnet18_big_model_0.41769_state.pt', map_location='cpu')) #
```



```


sub = pd.read_csv('C:/Users/COM/Desktop/nlsu/final_project_md/whale-categorization-playground/sample_submission.csv')
model1.eval()
model2.eval()
model3.eval()
model4.eval()
model5.eval()
model6.eval()
model7.eval()
model8.eval()
model9.eval()
for (data, target, name) in test_loader:
    data = data.cuda()
    output1 = model1(data)
    output2 = model2(data)
    output3 = model3(data)
    output4 = model4(data)
    output5 = model5(data)
    output6 = model6(data)
    output7 = model7(data)
    output8 = model8(data)
    output9 = model9(data)
    output1 = output1.cpu().detach().numpy()
    output2 = output2.cpu().detach().numpy()
    output3 = output3.cpu().detach().numpy()
    output4 = output4.cpu().detach().numpy()
    output5 = output5.cpu().detach().numpy()
    output6 = output6.cpu().detach().numpy()
    output7 = output7.cpu().detach().numpy()
    output8 = output8.cpu().detach().numpy()
    output9 = output9.cpu().detach().numpy()
    output = output1 + output2 + output3 + output4 + output5 + output6 + output7 + output8 + output9
    ensemble_output = torch.softmax(torch.tensor(output), dim=1).numpy()
    for i, (e, n) in enumerate(list(zip(ensemble_output, name))):
        sub.loc[sub['Image'] == n, 'Id'] = ' '.join([le.inverse_transform(e.argsort()[-5:][::-1])])
sub.to_csv('submission.csv', index=False)

```

양상블은 같은 구성의 9개의 모델로 구성하였고 각각의 모델의 output을 더하여 softmax를 취해 확률값으로 변환하여 진행하였습니다.

Leaderboard

YOUR RECENT SUBMISSION



submission.csv

Submitted by Jang_YoonSung · Submitted 2 minutes ago

Score: 0.46494

↓ Jump to your leaderboard position