Siamese Neural Network for One-shot Image Recognition (JMLR 2015)

Machine Learning Final Project

20195216 Lee Soeun

1 Before reading paper

데이터양이 많을수록 딥러닝 모델의 학습에 많은 도움이 된다. 그러나 현실을 잘 반영하는 데이터를 모으기는 쉽지 않다. 딥러닝 모델은 학습 시에 보지 못했던 형태를 마주했을 때 정확한 인식을 하기 어렵다. 반면에 사람은 적은 데이터로도 학습이 가능하다. 처음 보는 물체일지라도 몇 번 보면 그 물체를 구분할 수 있다. 이렇게 딥러닝 모델이 인간처럼 적은 양의 데이터만으로 학습하는 것을 Few-shot learning이라고 한다. Few-shot learning에서는 네트워크를 학습시킬 training set과 few shot learning을 수행하기 위한 support set, query data가 필요하다. support set은 training set에는 없는 새로운 클래스에 대한 이미지가 각 클래스 당 단 몇 장만 존재하는 data set이고, query data 또한 완전히 새로운 클래스에 대한 이미지 한 장으로, support set에 있는 이미지들과 비교된다. 여기서 정확히 해야 할 것은 적은 양의 데이터로 학습시키는 것이 아니라는 점이다. 모델을 학습시킬 때는 많은 양의 training set으로 학습시킨다. 학습 시 모델은 이미지들을 구분하는 법을 배운다. 그 후 test 시에 query data가 support set에서 존재하는 class 중 어떤 class와 같은지 예측하게 된다. 즉 test 시에 완전히 새로운 class에 대해 적은 양의 데이터로도 class 식별이 가능하다는 것이다. 이러한 few shot learning task를 'N-way K-shot 문제'라 하고 이때 N은 class의 수, K는 class 별 data의 수를 의미한다. N-way K-shot 문제에서 N과 K는 support set에 따라 결정된다. 만약 support set에 5개의 class가 있고 class 당 2장의 이미지가 있다면, 5-way 2-shot 문제가 된다. 따라서 few shot learning은 K가 작은 상황에서 모델을 학습시키는 것을 말한다. one shot learning은 few shot learning의 극단적인 예로, 각 class 당 단 하나의 data만 주어진 상태, 즉 K가 1인 상황에서 정확하게 예측하는 것을 말한다.

2 One shot learning

예를 들어, 한 아이에게 한 번도 본 적 없는 동물 이름이 달린 4장의 카드를 준다고 하자. 4장의 카드에는 사자, 호랑이, 기린, 코끼리가 있고 동물당 한 장의 카드만 존재한다. 이 4장의 카드를 본 아이는 이제 새로운 호랑이 사진을 보고 호랑이라고 말할 수 있게 된다. 아이는 본 적이 없던 동물 카드를 읽음으로써 스스로 학습을 한 것이다. 이것을 One shot learning에 대입해보면, 새로운 호랑이 사진을 query data라고 부른다. 그리고 4장의 카드 세트를 support set이라 하고, 아이가 동물당 한 장의 카드를 보고 학습하는 것을 one shot learning이라고 한다. Few shot learning의 N-way K-shot 문제와 같이, one shot learning에서는 K가 1인 N-way 1-shot 문제로, class 당 하나의 이미지가 있는 것을 뜻한다.

3 Abstract

좋은 feature를 학습하기 위해서는 계산 비용과 데이터가 많이 요구된다. 따라서, 각 class 당 하나의 example만으로 정확하게 예측하는 방법인 One-shot learning을 수행하고자 한다. Siamese neural network를 사용하여 input 간의 유사성을 측정하고, 처음 보는 class에 대해 추가적인 학습 과정 없이 좋은 성능을 낼 수 있도록 일반화하는 것이 목표이다.

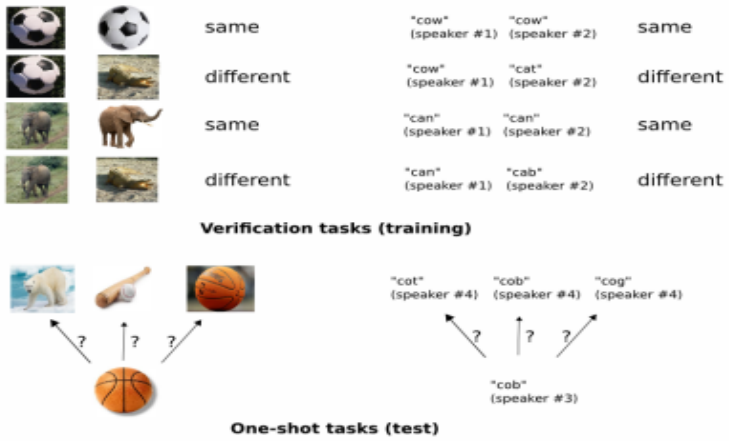


Figure 1: 1) same/different pairs를 구분하도록 모델을 학습 시킨다. 2) 검증을 위해 학습된 feature mapping을 기반으로 새로운 class를 평가하도록 일반화 한다.

One-shot image classification model을 만들기 위해, 첫 번째로 이미지 쌍의 class를 구분할 수 있는 neural network를 학습시키는 것을 목표로 한다. 입력으로 두 이미지가 들어가고, 이 두 이미지에 대한 유사도를 나타내는 값에 따라 input 쌍이 같은 class 인지 다른 class인지 식별하게 된다. neural network를 학습시킨 후에, 처음 보는 class가 어떤 class에 속하는지 예측하도록 한다. Figure 1에서는 농구공이 query data가 되고 북극곰, 야구 배트, 농구공이 support set이 된다. 농구공이 처음 보는 class일지라도 농구공 사진이 1장 주어졌을 때 어떤 class에 속하는지 예측할 수 있다. 이러한 task를 one-shot image recognition이라고 한다.

4 Siamese Neural Network

본 논문에서는 one-shot learning을 위해 Siamese neural network를 사용한다. 샴쌍둥이는 신체를 공유하고, 생김새도 비슷하다. Siamese network는 샴쌍둥이처럼 두 개의 네트워크로 구성되며 두 구조가 서로 닮았고 weight를 공유한다. Siamese network는 별개의 입력을 받아들이고, Loss function에 의해 결합하는 twin network 구조로 구성되며, 두 이미지 간의 유사도를 반환하는 네트워크이다. twin network의 parameter는 같고, 네트워크 구조가 대칭적인 것이 큰 특징이다. parameter가 같기 때문에, 비슷한 두 이미지를 넣었을 때 feature space에서 비슷한 위치로 mapping 되도록 한다.

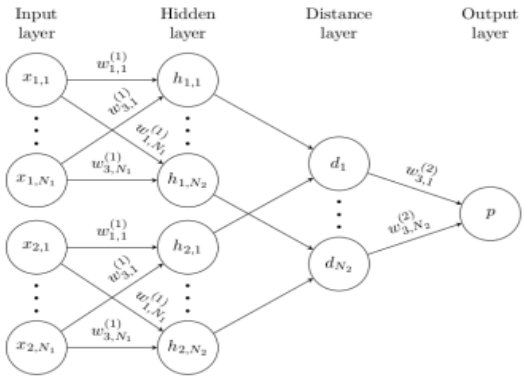


Figure 2: Figure 2. 본 그림은 2개의 hidden layer를 가진 간단한 Siamese network이다. 네트워크는 twin network를 구성하기 위해 상단과 하단에 복제되고 각 layer는 공유된 weight를 가진다.

논문에서는 twin network 간의 parameter가 tied 되었다고 표현하는데, 묶이지 않고 별개로 학습된다면 서로 다른 위치에 mapping 될 수 있다.

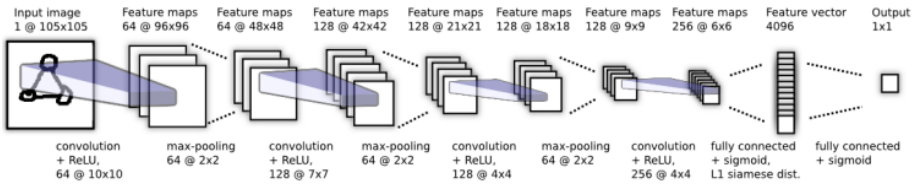


Figure 3: Siamese twin은 표현되지 않은 가장 좋은 성능의 구조. Siamese twin은 두 vector 간의 L1 distance가 계산되는 4096 unit fully-connected layer를 지난 후에 결합한다.

Network를 자세히 살펴보면 Convolution layer와 max pooling layer가 반복되고, 끝단에서 fully connected layer와 sigmoid를 거치게 된다. 여기까지만 본다면 평범한 network 구조겠지만, Siamese network는 똑같은 네트워크가 하나 더 존재한다. 마지막 convolution layer의 unit 들은 단일 vector로 flatten 된다. flatten 된 twin feature vector h1과 h2는 L1 distance를 통해 두 vector 사이의 거리가 계산되고, 0과 1 사이로 mapping 되는 sigmoid function과 결합한다. 즉, 두 네트워크를 통과하여 나온 두 output이 L1 distance를 통해 단일 값이 된다.

$$\sigma(\sum_j \alpha_j |\mathbf{h}^{(j)}_{1,L-1} - \mathbf{h}^{(j)}_{2,L-1}|)$$

식을 보면 두 feature vector간의 차를 구한 후 α_j를 곱한다. α_j 는 train 과정에서 자동으로 학습해주는 parameter로, 두 이미지가 같은지 다른지를 판별하기 위해 두 feature vector 간의 거리를 계산할 때 중요한 feature에 가중치를 주기 위한 parameter이다. 그 후 이 값에 sigmoid를 씌워 0과 1 사이의 값으로 만들어준다.

5 Implementing the Siamese network

- model class code

```
class Siamese(nn.Module):
    def __init__(self):
        super().__init__()

        # input: 1x105x105
        self.conv1 = nn.Conv2d(1, 64, 10)   # 64x96x96  --> 64x48x48
        self.conv2 = nn.Conv2d(64, 128, 7)  # 128x42x42 --> 128x21x21
        self.conv3 = nn.Conv2d(128, 128, 4) # 128x18x18 --> 128x9x9
        self.conv4 = nn.Conv2d(128, 256, 4) # 256x6x6

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(256*6*6, 4096)
        self.fc2 = nn.Linear(4096, 1)

        self.relu = nn.ReLU(True)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x1, x2):

        x1 = self.pool(self.relu(self.conv1(x1)))
        x1 = self.pool(self.relu(self.conv2(x1)))
        x1 = self.pool(self.relu(self.conv3(x1)))
        x1 = self.relu(self.conv4(x1))

        x2 = self.pool(self.relu(self.conv1(x2)))
        x2 = self.pool(self.relu(self.conv2(x2)))
        x2 = self.pool(self.relu(self.conv3(x2)))
        x2 = self.relu(self.conv4(x2))

        x1 = x1.view(x1.shape[0], -1)
        x2 = x2.view(x2.shape[0], -1)

        h1 = self.sigmoid(self.fc1(x1))
        h2 = self.sigmoid(self.fc1(x2))

        # L1 distance
        dist = torch.abs(h1-h2)

        p = self.sigmoid(self.fc2(dist))

        return p
```

논문에 나와 있는 모델 구조에 따라 작성한 Siamese network class이다. 총 4개의 convolution layer와 2개의 fully connected layer로 이루어져 있으며, convolution layer를 모두 거친 후 flatten 된 두 vector 사이의 L1 distance를 구해준다. 단일 값이 된 distance는 fully connected layer를 한 번 더 거치게 된다. 이는 중요한 feature에 대한 가중치인 α_j를 학습시키기 위함이다.

6 Experiment

본 논문에서는 character recognition에 대해 실험을 진행한다. 사용한 데이터 셋은 Omniglot 데이터 셋으로, 한국어와 같은 잘 알려진 언어와 잘 알려지지 않은 지역 방언까지 50가지의 언어로 구성되어 있다.

각 언어는 15개에서 40개의 글자(character)로 구성되어 있다. 또한 각 글자는 20명의 사람으로부터 그려져 글자당 20개의 데이터가 존재한다. 50가지의 언어는 background set과 evaluation set으로 나뉘는데, background set은 model을 개발하는 데에 사용되며 50가지의 언어 중 40가지 언어가 사용되고, evaluation set은 one-shot classification 성능을 평가하는 데에 사용되며 50가지의 언어 중 나머지 10가지 언어가 사용된다. background set은 다시 train과 validation set으로 나뉘고, evaluation set은 test set이 된다. 본 논문에서는 20-way one-shot에 대한 실험을 진행한다. 즉, 20개의 class가 존재하고 각 class는 한 장의 사진으로 구성된다. test

를 진행할 때, evaluation set에서 2명의 drawer인 A와 B를 선택하고 A와 B는 각각 20개의 data를 가진다. A의 20개의 data 중 하나가 test image가 되고, A에서 뽑은 test image를 B의 20개의 data와 각각 비교한다. 이때 A에서 뽑은 test image가 query data가 되고, B의 20개의 data가 support set이 된다. 20개의 쌍 중 가장 유사도가 가장 높은 쌍을 test image의 class로 식별하게 되는 것이다.

7 Implementing data set and data loader

- model class code

	A	B	C	D
1	image_path	language	character	image
2	./background/train/Alphabet_of_the_Magi₩character01₩0709_01.png	Alphabet_of_the_Magi	character01	0709_01.png
3	./background/train/Alphabet_of_the_Magi₩character01₩0709_02.png	Alphabet_of_the_Magi	character01	0709_02.png
4	./background/train/Alphabet_of_the_Magi₩character01₩0709_03.png	Alphabet_of_the_Magi	character01	0709_03.png
5	./background/train/Alphabet_of_the_Magi₩character01₩0709_04.png	Alphabet_of_the_Magi	character01	0709_04.png
6	./background/train/Alphabet_of_the_Magi₩character01₩0709_05.png	Alphabet_of_the_Magi	character01	0709_05.png
7	./background/train/Alphabet_of_the_Magi₩character01₩0709_06.png	Alphabet_of_the_Magi	character01	0709_06.png
8	./background/train/Alphabet_of_the_Magi₩character01₩0709_07.png	Alphabet_of_the_Magi	character01	0709_07.png
9	./background/train/Alphabet_of_the_Magi₩character01₩0709_08.png	Alphabet_of_the_Magi	character01	0709_08.png
10	./background/train/Alphabet_of_the_Magi₩character01₩0709_09.png	Alphabet_of_the_Magi	character01	0709_09.png
11	./background/train/Alphabet_of_the_Magi₩character01₩0709_10.png	Alphabet_of_the_Magi	character01	0709_10.png
12	./background/train/Alphabet_of_the_Magi₩character01₩0709_11.png	Alphabet_of_the_Magi	character01	0709_11.png
13	./background/train/Alphabet_of_the_Magi₩character01₩0709_12.png	Alphabet_of_the_Magi	character01	0709_12.png
14	./background/train/Alphabet_of_the_Magi₩character01₩0709_13.png	Alphabet_of_the_Magi	character01	0709_13.png
15	./background/train/Alphabet_of_the_Magi₩character01₩0709_14.png	Alphabet_of_the_Magi	character01	0709_14.png

background set과 evaluation set에 대하여 data를 load 할 때 편하도록 모든 이미지를 image path, language, character, image 열에 각각 저장해두었다. image_path에는 이미지의 전체 경로, language는 50가지의 언어, character는 언어당 존재하는 15에서 40개의 글자, image는 이미지 이름에 해당한다. 이때, character가 class가 된다.

- train dataset code

```
class MyOmniglot_dataset():
    def __init__(self, csv, transform=None):
        self.csv = csv
        self.transforms = transform

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

    def __getitem__(self, index):
        ridx = np.random.choice(len(self.csv))

        image1 = self.csv['image_path'][ridx]
        lang1 = self.csv['language'][ridx]
        chr1 = self.csv['character'][ridx]

        same_condition = (self.csv.language == lang1) & (self.csv.character == chr1)
        diff_condition = (self.csv.language != lang1) | (self.csv.character != chr1)

        self.same = self.csv[same_condition].reset_index()
        self.diff = self.csv[diff_condition].reset_index()

        if index % 2 == 0:
            same_idx = np.random.choice(len(self.same))
            image2 = self.same['image_path'][same_idx]
            label = 1

        else:
            diff_idx = np.random.choice(len(self.diff))
            image2 = self.diff['image_path'][diff_idx]
            label = 0

        image1 = cv2.imread(image1, cv2.COLOR_BGR2GRAY)
        image2 = cv2.imread(image2, cv2.COLOR_BGR2GRAY)

        image1 = np.expand_dims(image1, 2)
        image2 = np.expand_dims(image2, 2)

        if self.transforms != None:
            augmentation = self.transforms(image=image1, image2=image2)
            img1 = augmentation['image']
            img2 = augmentation['image2']

        return img1, img2, label
```

dataset class에서는 만들어놓은 csv를 받게 된다. train data 중 하나를 random 으로 골라 image 1로 설정해주고 image 1의 언어와 글자도 lang1, chr1에 각각

저장해둔다. image 2를 뽑기 위해서는 다음과 같은 과정을 거친다.

1. image 2를 뽑을 때 언어와 글자가 image 1과 같은 경우 즉, 같은 class일 조건을 만족한다면 same_condition에 저장, 다른 class일 경우에는 diff_condition에 boolean 형으로 저장한다.

2. same_condition과 diff_condition을 train_csv에 적용해주면 image 1과 같은 class인 이미지의 경로와 다른 class인 이미지의 경로가 same, diff에 각각 저장된다.

3. 이렇게 같은 이미지와 다른 이미지를 나눈 후, getitem의 index를 기준으로 index가 짝수일 경우와 홀수일 경우로 나눈다. 이는 data loader에서 이미지를 뽑을 때 image 쌍이 같은 class인지 다른 class인지에 대한 비율을 50:50으로 맞추기 위함이다. index가 짝수일 때는 두 이미지를 같은 class 쌍으로 만들어주고, 홀수일 경우에는 다른 class 쌍으로 만들어준다.

4. 또한, 같은 class일 경우엔 label 값을 1로, 다른 class일 경우에는 label 값을 0으로 설정하여 최종적으로 image 1, image 2, label을 반환해준다.

- Loss function & optimization

```
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=1e-2, momentum=0.9)
# optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay = 0.01) # 1e-4
# scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.99)
```

해당 문제는 같다(1)와 다르다(0)를 판별하기 때문에 binary cross entropy loss를 사용하였고 optimizer는 adam과 SGD+momentum 두 개를 실험해 본 결과 SGD+momentum을 사용했을 때 성능 향상이 있었기 때문에 SGD+momentum을 사용하였다. scheduler를 사용하여 learning rate를 크게 주고 학습을 진행해 보았지만 scheduler를 사용하지 않았을 때의 성능이 더 좋았다.

optimal hyperparameter setting	
batch size	128
optimizer	SGD+momentum
learning rate	1e-2

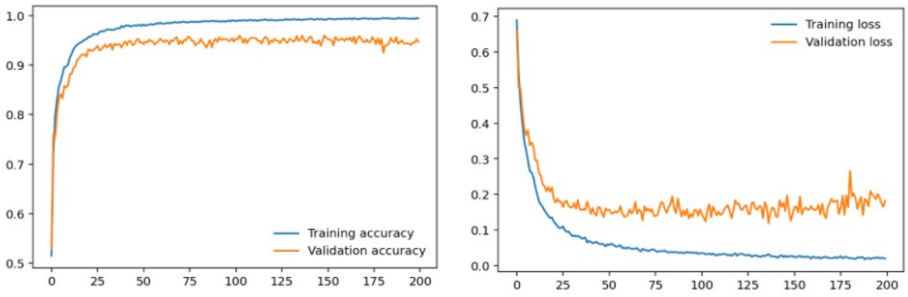
8 Implementing train, validation code

- train & validation

```
epoch = 200
checkpoint = 0
# model.apply(weight_init)

train_loss = torch.zeros(epoch)
valid_loss = torch.zeros(epoch)
train_acc = torch.zeros(epoch)
valid_acc = torch.zeros(epoch)
valid_loss_min = np.Inf

for e in np.arange(epoch):
    model.train()
    for data1, data2, labels in tqdm(train_loader):
        data1, data2, labels = data1.to(device), data2.to(device), labels.to(device)
        labels = labels.unsqueeze(1).type(torch.float32)

        optimizer.zero_grad()
        logits = model(data1, data2)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()

        train_loss[e] += loss.item()
        equals = (logits > 0.5) == labels
        train_acc[e] += torch.mean(equals.type(torch.float)).detach().cpu()

    train_loss[e] /= len(train_loader)
    train_acc[e] /= len(train_loader)

    model.eval()
    with torch.no_grad():
        for data1, data2, labels in tqdm(valid_loader):
            data1, data2, labels = data1.to(device), data2.to(device), labels.to(device)
            labels = labels.unsqueeze(1).type(torch.float32)

            logits = model(data1, data2)
            loss = criterion(logits, labels)

            valid_loss[e] += loss.item()
            equals = (logits > 0.5) == labels
            valid_acc[e] += torch.mean(equals.type(torch.float)).detach().cpu()

    valid_loss[e] /= len(valid_loader)
    valid_acc[e] /= len(valid_loader)

    scheduler.step()

    print('Epoch: {} \tTraining acc: {:.6f} \tTraining Loss: {:.6f} \tValidation acc: {:.6f} \
          \tValidation Loss: {:.6f} \tlr: {:.7f} \tno_update : {}'.format(
          e, train_acc[e], train_loss[e], valid_acc[e], valid_loss[e],
          optimizer.param_groups[0]["lr"], checkpoint))

    if valid_loss[e] <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss[e]))

        torch.save(model.state_dict(), f'model/model_best_SGD_lr1e-2_scheduler.pt')
        valid_loss_min = valid_loss[e]

        checkpoint = 0
    else:
        checkpoint += 1
```

학습 시에는 validation loss를 기준으로 validation loss가 가장 작을 때 model을 저장하도록 설정하였다. 또한 checkpoint로 validation loss가 줄어들고 있는지 확인하였다. epoch를 200으로 설정하여 학습시켰는데 epoch가 50이 넘어가면 validation loss가 잘 줄어들지 않았다.

- train, validation accuracy & loss graph



x축은 epoch, y축은 왼쪽 graph은 accuracy 오른쪽 graph는 loss를 나타낸다.

9 Result

본 논문에서는 20-way one-shot 문제로 test를 진행하였다. 20개의 클래스당 1장의 이미지만 존재하게 된다. batch size를 20으로 설정해두어 20개의 class에 대한 image를 가져오도록 하였다. 이미지 2장이 input으로 들어오는데 한 장은 test image가 되고 다른 한 장은 support set에서 뽑은 이미지로 image 1과 같은 class일 수도, 다른 class일 수도 있다. test image는 support set에 있는 20개의 이미지 모두와 같은 class 인지 다른 class인지 비교하게 된다. 따라서 한 batch 당 20개의 쌍이 존재하게 되고 이 20개의 쌍 중에 유사도가 가장 높은 쌍을 통해 image 1과 image 2가 같은 class임을 알 수 있다.

- test dataset code

```
class test_dataset():
    def __init__(self, csv, batch_size, transform=None):
        self.csv = csv
        self.batch = batch_size
        self.transforms = transform

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

    def __getitem__(self, index):
        ridx = np.random.choice(len(self.csv))
        image1 = self.csv['image_path'][ridx]
        lang1 = self.csv['language'][ridx]
        chr1 = self.csv['character'][ridx]

        same_condition = (self.csv.language == lang1) & (self.csv.character == chr1)
        diff_condition = (self.csv.language != lang1) | (self.csv.character != chr1)

        self.same = self.csv[same_condition].reset_index()
        self.diff = self.csv[diff_condition].reset_index()

        if index % batch_size == 0:
            same_idx = np.random.choice(len(self.same))
            image2 = self.same['image_path'][same_idx]
            while image1.split('\\')[2] == image2.split('\\')[2]:
                same_i = np.random.choice(len(self.same))
                image2 = self.same['image_path'][same_i]

        else:
            diff_i = np.random.choice(len(self.diff))
            image2 = self.diff['image_path'][diff_i]
            while image1.split('\\')[2] == image2.split('\\')[2]:
                diff_i = np.random.choice(len(self.diff))
                image2 = self.diff['image_path'][diff_i]

        image1 = cv2.imread(image1, cv2.COLOR_BGR2GRAY)
        image2 = cv2.imread(image2, cv2.COLOR_BGR2GRAY)

        image1 = np.expand_dims(image1, 2)
        image2 = np.expand_dims(image2, 2)

        if self.transforms != None:
            augmentation = self.transforms(image=image1, image2=image2)
            img1 = augmentation['image']
            img2 = augmentation['image2']

        return img1, img2
```

test dataset은 train dataset과 마찬가지로 image 1을 정해두고 image 1과 같은 class, 다른 class로 나눈다. test에서는 image 1이 test data가 되고 20개의 class에 대한 image set은 support set이 된다. test dataset에서는 20으로 설정해둔 batch_size로 index를 나누게 되는데, 이는 0을 포함하여 20의 배수일 때만, 즉 20개의 이미지 쌍에 대해 첫 번째 쌍만 같은 class로 설정하고 나머지 쌍은 다른 class로 설정한다. while 문은 완전히 같은 이미지를 뽑는 것을 방지하기 위해 이미지 이름이 같을 때 image 2를 다시 뽑도록 설정해주었다.

- test code

```
correct = 0
model.eval()
with torch.no_grad():
    for data1, data2 in tqdm(test_loader):

        data1, data2 = data1.to(device), data2.to(device)

        logits = model(data1, data2)
        logits = torch.argmax(logits)

        if logits == 0:
            correct += 1

test_acc = (100*correct)/len(test_loader)
print(f"test accuracy: {correct}/{len(test_loader)} ({test_acc:.2f}%)")
```

```
100% ████████████████████████████████████████ 352/352 [00:42<00:00, 8.51it/s]
test accuracy: 321/352 (91.19%)
```

test code를 짜는 부분은 오픈 소스의 도움을 받았다. 20개의 이미지 쌍 중 제일 첫 번째 쌍만 같은 class로 둔다는 점이 모든 오픈 소스에서의 공통점이었다. 같은 class 쌍에서 가장 높은 유사도가 나와야 정답을 맞힌 것이므로, 20개의 model output들에 argmax를 씌웠을 때 첫 번째 쌍의 index인 0이 나와야 정답을 맞히는 것이다. 따라서 model output에 argmax를 씌운 값이 0일 때 correct 변수에 1씩 더해준 후 test loader 길이로 나누어 정확도를 구하였다. 최종적으로 test에서 91.19%의 정확도를 얻었다.