Custom dataset 활용 및 학습을 위한

Pytorch 딥러닝 빠르게 시작하기

ISL

안재원

CONTENTS

- Introduction
- Dataset 만들기
- Network 만들기
- 학습하기
- 결과

[참고자료]
- Pytorch tutorials 페이지
- Pytorch docs 페이지

Introduction

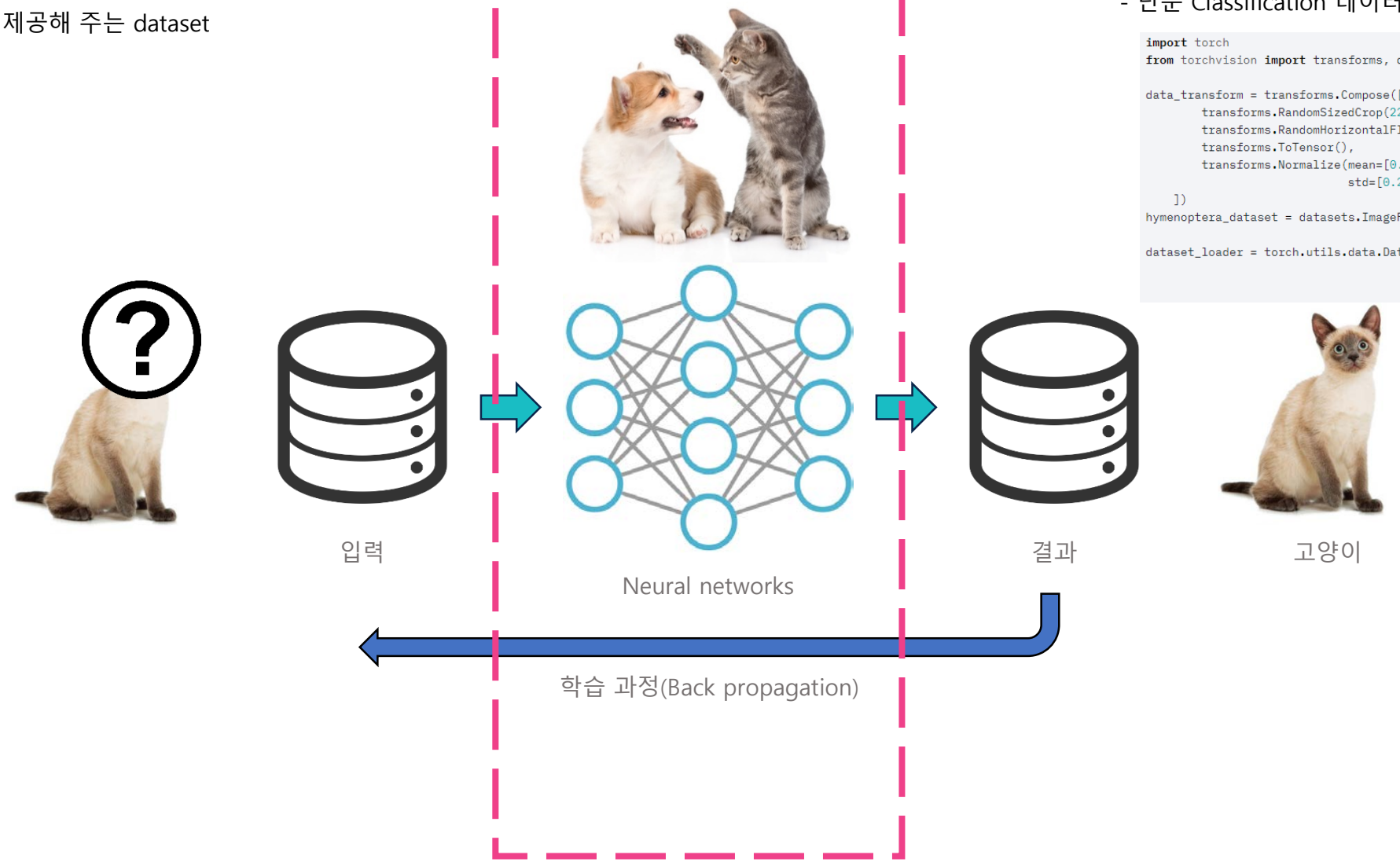
학습을 위해 필요 한 것들

- Pytorch에서 기본적으로 제공해 주는 dataset

The following datasets are available:

Datasets

- MNIST
- Fashion-MNIST
- KMNIST
- EMNIST
- QMNIST
- FakeData
- COCO
  - Captions
  - Detection
- LSUN
- ImageFolder
- DatasetFolder
- ImageNet
- CIFAR
- STL10
- SVHN
- PhotoTour
- SBU
- Flickr
- VOC
- Cityscapes
- SBD
- USPS
- Kinetics-400
- HMDB51
- UCF101



- 단순 Classification 데이터 처리하기

```
import torch
from torchvision import transforms, datasets

data_transform = transforms.Compose([
        transforms.RandomSizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
hymenoptera_dataset = datasets.ImageFolder(root='hymenoptera_data/train',
                                           transform=data_transform)
dataset_loader = torch.utils.data.DataLoader(hymenoptera_dataset,
                                             batch_size=4, shuffle=True,
                                             num_workers=4)
```

- 쉽게 접하는 내용들은 보통 네트워크와 학습에 관한 부분을 담고 있다.

Introduction

학습을 위해 필요 한 것들

- 내가 원하는 데이터로 학습을 하고 싶다면?

The following datasets are available:

Datasets

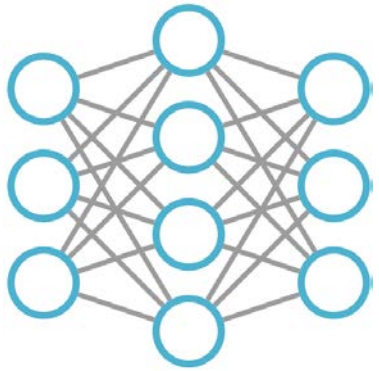
- MNIST
- Fashion-MNIST
- KMNIST
- EMNIST
- QM[illegible]T
- Fake[illegible]a
- COCO
  - Cap[illegible]ns
  - Dete[illegible]on
- LSUN
- ImageFolde[illegible]
- DatasetFold[illegible]
- ImageNet
- CIFAR
- STL10
- SVHN
- Photo[illegible]r
- SBU
- Flick[illegible]
- VOC
- Cityscapes
- SBD
- USPS
- Kinetics-400
- HMDB51
- UCF101

-1	0	+1
-2	0	+2
-1	0	+1

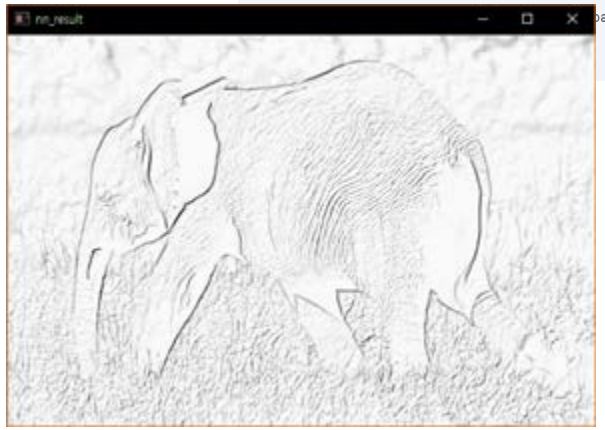
Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy



Neural networks



학습 과정(Back propagation)

- 단순 Classification 데이터 처리하기

```
import torch
from torchvision import transforms, datasets

data_transform = transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ])
hymenoptera_dataset = datasets.ImageFolder('hymenoptera_data/train',
                                           transform=data_transform)
dataloader = torch.utils.data.DataLoader(hymenoptera_dataset,
                                         batch_size=4, shuffle=True,
                                         num_workers=4)
```

그래서 이번 세미나는

- Custom dataset class 만드는 방법
- 네트워크 구성하는 코드 작성 방법
- 학습하는 코드 작성 방법

Dataset 만들기

Custom datasets

- Pytorch에서 기본적으로 제공해 주는 dataset

The following datasets are available:

Datasets

- MNIST
- Fashion-MNIST
- KMNIST
- EMNIST
- QMNIST
- FakeData
- COCO
  - Captions
  - Detection
- LSUN
- ImageFolder
- DatasetFolder
- ImageNet
- CIFAR
- STL10
- SVHN
- PhotoTour
- SBU
- Flickr
- VOC
- Cityscapes
- SBD
- USPS
- Kinetics-400
- HMDB51
- UCF101

- 기본 dataset 사용법

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

- 학습을 위한 입력 데이터의 변환 정보.
- 일단, toTensor()는 반드시 필요하다.(뒤에 설명)

Train set

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
```

데이터 저장 위치

Train, Test 구분

다운로드

변환 정보

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
```

Batch 크기

기타 설정

Test set

```
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)
```

- Custom dataset을 만드는 것은?

DataLoader 함수에 입력될 Datasets을 만드는 것

<Datasets의 조건>

1. Dataset 'Class'로 생성 해야됨.
2. len(dataset) 함수로 dataset의 크기를 받을 수 있어야 함.
3. dataset[i]로 i번째 dataset에 접근 할 수 있어야 함.
4. 필요한 transform class를 구현 해야 함.

Dataset 만들기

Dataset class

- class EdgeTestDataSet()

```
import torch
import torchvision
import torchvision.transforms as transforms

from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
```

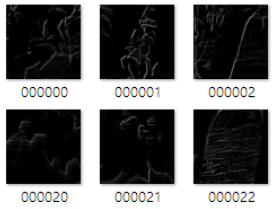
- Torch.utils.data의 Dataset class 상속

```
class EdgeTestDataSet(Dataset):
    """Edge detection Test DataSet."""

    def __init__(self, source_image_dir, result_image_dir, transform=None):

        self.source_file_names = os.listdir(source_image_dir)
        #bmp만 읽기
        #self.source_file_names_bmp = [file for file in source_file_names if file.endswith(".bmp")]
        self.source_image_dir = source_image_dir
        self.result_image_dir = result_image_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        #파일 위치 만들기
        source_img_name = os.path.join(self.source_image_dir,self.source_file_names[idx])
        result_img_name = os.path.join(self.result_image_dir,self.source_file_names[idx])

        #파일 읽기
        source_image = io.imread(source_img_name)
        result_image = io.imread(result_img_name)

        #저장
        sample = {'source': source_image, 'result': result_image}

        #정해진 transform에 맞춰 변환
        if self.transform:
            sample = self.transform(sample)

        return sample
```

- 클래스 생성자에서 필요한 정보 저장.
  1. source_file_names : 사용할 파일명 목록
  2. source_image_dir : 입력 파일 위치
  3. result_image_dir : 정답 파일 위치
  4. transform : 입력 파일 변환 정보.

<Input>

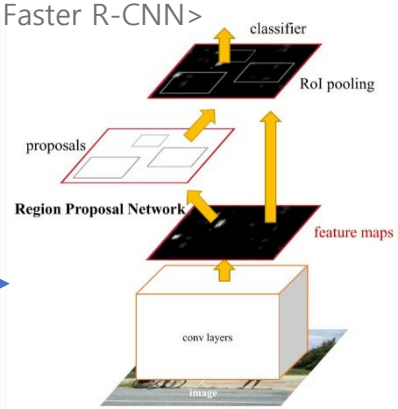
<Output>



- __len__()과 __getitem__()을 재정의

- __getitem__() 재정의
  - 입력된 idx 값에 해당하는 값 출력
  - 결국, 원하는 값들이 정리된 형태로 반환 되도록 한다.
  - 입력과 출력이 1:1 관계일 필요가 없다.

<Faster R-CNN>



<Datasets의 조건>

1. Dataset 'Class'로 생성 해야됨.
2. len(dataset) 함수로 dataset의 크기를 받을 수 있어야 함.
3. dataset[i]로 i번째 dataset에 접근 할 수 있어야 함.
4. 필요한 transform class를 구현 해야 함.

Dataset 만들기

Transform class

- Transform class

```
        #저장
        sample = {'source': source_image, 'result': result_image}

        #정해진 transform에 맞춰 변환
        if self.transform:
            sample = self.transform(sample)

        return sample
```

- 데이터를 학습에 적합한 형태로 변환하는 과정
- torchvision.transforms에 기본적으로 제공해주지만, 원하는 변환이 없으면 따로 생성해 준다.
  - torchvision.transforms.CenterCrop()
  - torchvision.transforms.ColorJitter()
  - torchvision.transforms.Grayscale()
  - torchvision.transforms.Pad()
  - torchvision.transforms.RandomAffine()
  - etc....
- 또한, torchvision.transforms PIL Image로 동작하거나 Tensor Image로 동작하기 때문에 순서에 맞춰 transform function을 사용해야 한다.

- Custom Transform class 예시(Grayscale)

```
class Grayscale(object):
    """Convert to Grayscale"""

    def __call__(self, sample):
        source, result = sample['source'], sample['result']

        #source 확인
        if source.ndim == 3:
            source = cv.cvtColor(source, cv.COLOR_BGR2GRAY)

        #result 확인
        if result.ndim == 3:
            result = cv.cvtColor(result, cv.COLOR_BGR2GRAY)

        return {'source': source,
                'result': result}
```

<Datasets의 조건>
1. Dataset 'Class'로 생성 해야됨.
2. len(dataset) 함수로 dataset의 크기를 받을 수 있어야 함.
3. dataset[i]로 i번째 dataset에 접근 할 수 있어야 함.
4. 필요한 transform class를 구현 해야 함.

Dataset 만들기

Transform class

- ToTensor Transform class
  - Tensor가 데이터를 다루는 특성에 의해 반드시 필요한 과정

```python
class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample):
        source, result = sample['source'], sample['result']

        # swap color axis because
        # numpy image: H x W x C
        # torch image: C X H X W

        #source 3차원으로 변경.
        if source.ndim == 2:
            source = np.expand_dims(source,axis=2)

        #result 3차원으로 변경.
        if result.ndim == 2:
            result = np.expand_dims(result,axis=2)

        source = source.transpose((2, 0, 1))
        result = result.transpose((2, 0, 1))

        return {'source': torch.from_numpy(source),
                'result': torch.from_numpy(result)}
```

- 앞서 과정에서 불러온 영상의 저장 순서와 Tensor가 다루는 영상의 저장 순서가 다르다.
  - Transpose를 이용해 데이터의 저장 형태 변경.
  - (0,1,2) -> (2,0,1) 순서로 변경.

- 채널이 1개인 영상도 채널 정보가 있어야 한다.
  - (500,500)의 크기를 갖는 1채널 영상의 크기는 (500,500)이 아니라. (500,500,1)로 저장 되어 야 한다.

- Tensor로 변환해 저장.

<Datasets의 조건>
1. Dataset 'Class'로 생성 해야됨.
2. len(dataset) 함수로 dataset의 크기를 받을 수 있어야 함.
3. dataset[i]로 i번째 dataset에 접근 할 수 있어야 함.
4. 필요한 transform class를 구현 해야 함.

Dataset 만들기

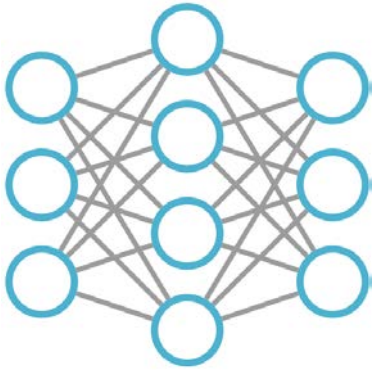
Custom data load

- 학습 데이터 불러 오기 예시

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
```

- Custom 학습 데이터 불러 오기 예시

```
#데이터 생성
transform = transforms.Compose(
    [Grayscale(),
     Normalize(-1.,1.),
     ToTensor()
    ]
     )

trainset = EdgeTestDataSet(source_image_dir='./data_set/inputs',
                           result_image_dir='./data_set/labels',
                           transform = transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
```



Neural networks

- DataLoader에서 출력되는 데이터의 형태는 [Batch_size, 입력 영상의 채널, 영상 높이, 영상 너비] 이다.
- 즉, 학습이 끝나고 테스트를 할 때도 이 형식에 맞춰 4개 상의 데이터를 사용해야 한다.

Network 만들기

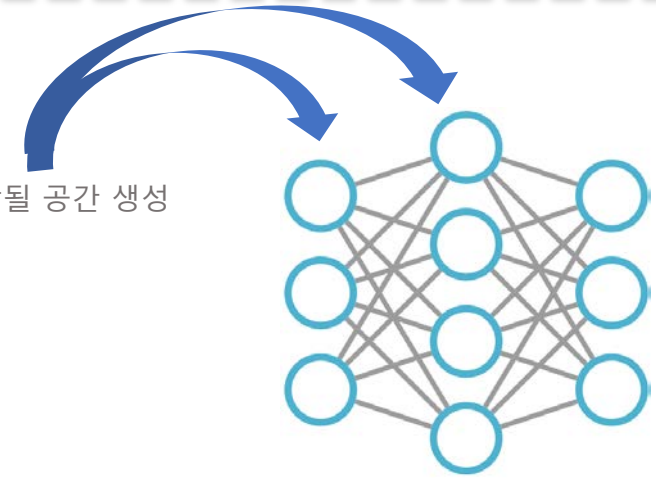
Network class

- Network 클래스 생성하기

```
import torch.nn as nn
import torch.nn.functional as F
```

```
#######################################################################
#네트워크 구성
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # conv1
        # input : 1 channel / output : 4 channel / conv : 3x3
        self.conv1 = nn.Conv2d(1, 4, 3)
        # conv2
        # # input : 4 channel / output : 1 channel / conv : 1x1
        self.conv2 = nn.Conv2d(4, 1, 1)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.conv2(x)
        return x
```

- 각 레이어가 저장될 공간 생성



Neural networks

-1	0	+1	+1	+2	+1
-2	0	+2	0	0	0
-1	0	+1	-1	-2	-1

Gx Gy

- 각 레이어가 어떻게 구성되어 있는가에 해당하는 정보.

학습하기

Training

```
#################################################################################
#네트워크 로드
net = Net()


print('<Network feature>')
print(net)

#################################################################################
#Define a Loss function and optimizer
criterion = nn.L1Loss()
optimizer = optim.SGD(net.parameters(), lr=0.05, momentum=0.9)

#################################################################################
#학습
for epoch in range(2): # 전체 과정 2번 반복
    running_loss = 0.0
    i = 0
    for i, data in enumerate(trainloader, 0): #데이터 로드 하면서 아래 과정 반복
        # get the inputs
        #inputs, result = data
        inputs, result = data['source']          ,data['result']

        # zero the parameter gradients
        optimizer.zero_grad()

        #포워드
        outputs = net(inputs)
        #로스 계산
        loss = criterion(outputs, result)
        #백워드
        loss.backward()
        #optimize
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 100 == 99:    # print every 20 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 100))
            running_loss = 0.0

        i = i+1
```

- 학습을 위한 코드 구성

1. 학습에 사용할 데이터 불러오기
2. Optimizer parameter 초기화
   - Gradient 정보.
3. 네트워크 출력 구하기
4. Loss 구하기
5. Network parameter 업데이트

- Network 불러오기

- Loss function 선택하기
  - torch.nn.L1Loss()
  - torch.nn.MSELoss()
  - torch.nn.CrossEntropyLoss()
  - torch.nn.CTCLoss()
  - torch.nn.NLLLoss()
  - torch.nn.PoissonNLLLoss()
  - etc..

- Optimizer선택하기
  - torch.optim.Adam()
  - torch.optim.RMSprop()
  - torch.optim.SGD()
  - etc..

학습하기

Over-view

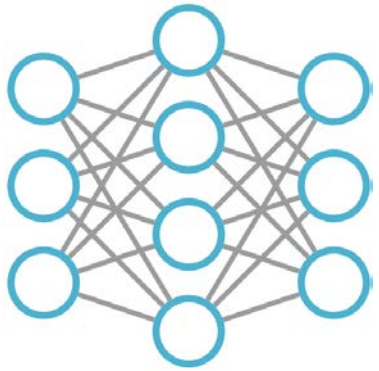
```python
class EdgeTestDataSet(Dataset):
    """Edge detection Test DataSet."""

    def __init__(self, source_image_dir, result_image_dir, transform=None):

        self.source_file_names = os.listdir(source_image_dir)
        #bmp만 읽기
        #self.source_file_names_bmp = [file for file in source_file_names if file.endswith(".bmp")]
        self.source_image_dir = source_image_dir
        self.result_image_dir = result_image_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        #파일 위치 만들기
        source_img_name = os.path.join(self.source_image_dir,self.source_file_names[idx])
        result_img_name = os.path.join(self.result_image_dir,self.source_file_names[idx])

        #파일 읽기
        source_image = io.imread(source_img_name)
        result_image = io.imread(result_img_name)

        #저장
        sample = {'source': source_image, 'result': result_image}

        #정의된 transform에 맞춰 변환
        if self.transform:
            sample = self.transform(sample)

        return sample
```

```python
class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample):
        source, result = sample['source'], sample['result']

        # swap color axis because
        # numpy image: H x W x C
        # torch image: C X H X W

        #source 3차원으로 변경.
        if source.ndim == 2:
            source = np.expand_dims(source,axis=2)

        #result 3차원으로 변경.
        if result.ndim == 2:
            result = np.expand_dims(result,axis=2)

        source = source.transpose((2, 0, 1))
        result = result.transpose((2, 0, 1))

        return {'source': torch.from_numpy(source),
                'result': torch.from_numpy(result)}
```

input

```python
#데이터 생성
transform = transforms.Compose(
    [Grayscale(),
     Normalize(-1.,1.),
     ToTensor()
    ]
    )

trainset = EdgeTestDataSet(source_image_dir='./data_set/inputs',
                           result_image_dir='./data_set/labels',
                           transform = transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)
```

```python
##################################################################
#네트워크 구성
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # conv1
        # input : 1 channel / output : 4 channel / conv : 3x3
        self.conv1 = nn.Conv2d(1, 4, 3)
        # conv2
        # # input : 4 channel / output : 1 channel / conv : 1x1
        self.conv2 = nn.Conv2d(4, 1, 1)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.conv2(x)
        return x
```



Neural networks

nn_result

```python
##################################################################################
#Define a loss function and optimizer
criterion = nn.L1Loss()
optimizer = optim.SGD(net.parameters(), lr=0.05, momentum=0.9)

##################################################################################
#학습
for epoch in range(2): # 전체 과정 2번 반복
    running_loss = 0.0
    i = 0
    for i, data in enumerate(trainloader, 0): #데이터 로드 하면서 아래 과정 반복
        # get the inputs
        #inputs, result = data
        inputs, result = data['source'].to(device),data['result'].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        #포워드
        outputs = net(inputs)
        #로스 계산
        loss = criterion(outputs, result)
        #백워드
        loss.backward()
        #optimize
        optimizer.step()
```

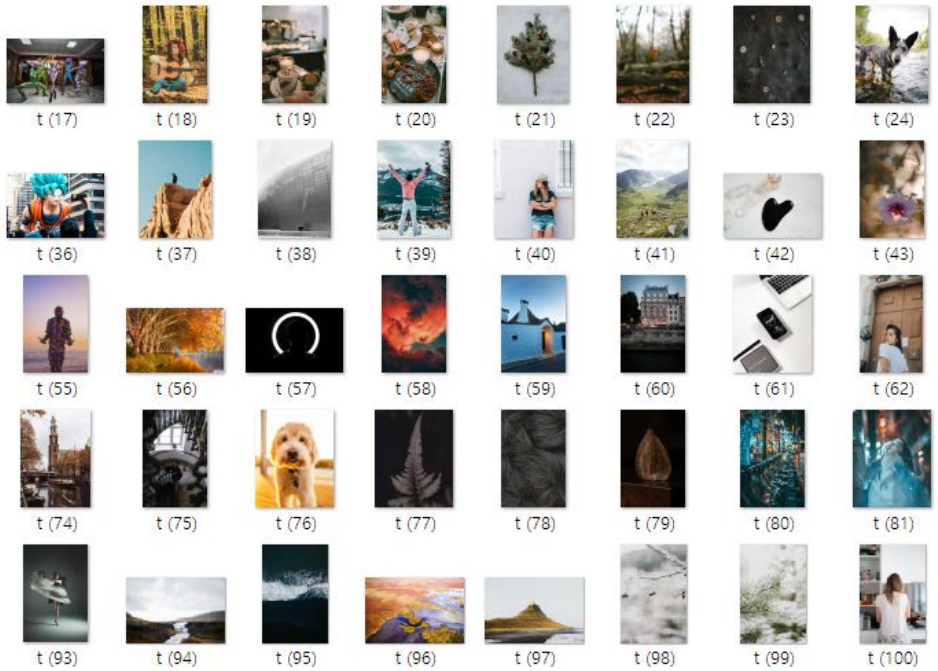
결과

학습 결과

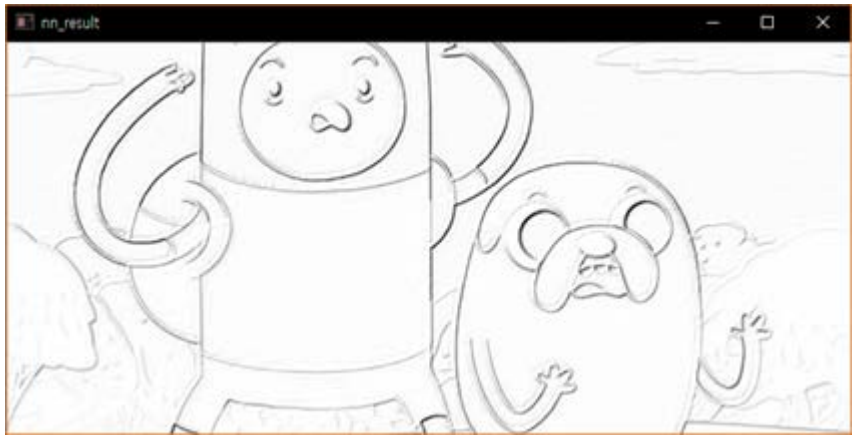
- unsplash.com에 올라온 사진을 이용해 학습

  - 고화질 영상을 (500,500)으로 잘라 사용.
  - 총 15,018장의 영상 사용.

  - 정답은 각 잘린 영상의 sobel 결과 영상을 사용.
  - 정답 영상의 크기는 (498,498)



- 학습 결과



Q & A