


Project Portfolio

Computer Vision 개발 지원자

김민정 입니다 .

Contact

 | 010-7263-1963

 | kmjng0712@gmail.com

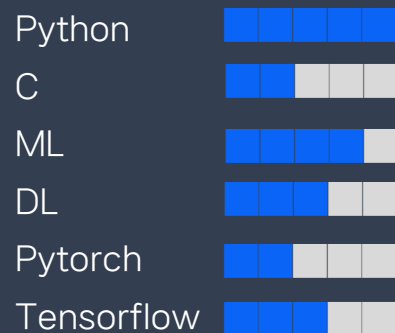
 | <https://github.com/Kmjng>

Contents

1. 이력사항 (p.3 ~ p.6)
2. 프로젝트 (p.4 ~ 14)
3. 상세 코드 (p.15 ~ 25)
4. 레퍼런스 (p.26)

[1] 이력사항

유관 스킬



자격증

2024 빅데이터분석기사
2024 SQLD
2024 ADsP
2020 MOS Master

학력사항

2021.03 ~ 2023.02

- 건국대학교 일반대학원 물리학과 (석사) 졸업

2018.03 ~ 2021.02

- 건국대학교 물리학과 (학사) 졸업

2015.03 ~ 2018.02

- 한성대학교 전자정보공학과 중퇴

2012.03 ~ 2015.02

- 판교고등학교 졸업

경력사항

* 비전공

2023.03 ~ 2023.12

- 에이배리스터컴퍼니(주) 석사후 연구원

수료 및 활동

2024.02 ~ 2024.08

- ITWILL(국비 교육) 빅데이터 융합 머신러닝 전문가 양성과정
-수료 및 표창 (최우수)

상세 스킬

* 딥러닝 프레임 워크

머신러닝 교육과정 중 [Tensorflow](#)을 기반으로 딥러닝을 공부했으며, 이외에 연구 및 활용도가 더 높은 [Pytorch](#)를 채택해 개인적으로 공부하여 프로젝트를 진행했습니다.

* Python

[기본적 문제해결](#), [데이터 분석](#) 및 [기계학습 모델링](#)을 위한 Python 스킬 보유

* C

편입 전 전공(전자정보공학)에서 수강했으며, 절차적 프로그래밍 해석하는 수준의 스킬 보유

* 문서 작업 능력

석사 과정 중 [Microsoft Office](#) 및 [한글](#)을 자주 사용했으며, 코드 관리는 [Github](#) 및 [Git](#)을 주로 활용했습니다.

성격의 장단점

장점

성취 지향적/자기주도적/체력

배운 것을 적용하고 응용하는 것을 좋아합니다.

새로움의 발견은 주도적인 생각과 **궁금증**으로부터 나온다고 생각합니다.
머신러닝 교육 과정동안, 두번의 팀프로젝트에서 기존에 없던 지표를 만들려고 했으며
수료 당시 현직자 튜터에게 특허출허 추천과 좋은 평가를 받았습니다.

컴퓨터비전에 대한 관심으로 개인프로젝트를 주도적으로 진행해 기존 모델링에 추가
적으로 적용할만한 기술을 Docs와 서적을 통해 공부하고 적용해가며 전문 지식을 쌓
고 있습니다.

단점

많은 생각

어떤 것을 접했을 때 많은 궁금증과 생각으로 정리가 안될 때가 있습니다.
집중을 효율적으로 하기 위해 스케줄러를 작성하고, 개인 노선에 업무 순서와 진척도
를 정리하며 머릿속을 정리합니다.

이전 경력 (~ 2년 6개월)

* 비전공

업무 내용

- | | |
|------------------|---|
| 1. 수업 조교 활동 | <ul style="list-style-type: none">21. 2학기 - 전산물리 조교 (Matlab)22. 1학기 - 컴퓨터 프로그래밍 조교 (Python, Deep Learning) |
| 2. 실험 물리학 | <ul style="list-style-type: none">배리스터 (반도체) 소자 공정 및 연구 |
| 3. 지도교수 창업 회사 업무 | <ul style="list-style-type: none">국가과제 관리 (초기창업패키지/서울기술연구원 과제)R&D (배리스터 센서 양산 단계별 공정 업체 협업) |

업무 성과

- | | |
|----------------|--|
| - SCI 논문 공동 저술 | <ul style="list-style-type: none">Choi, Inchul, et al. Advanced Electronic Materials 8.12 (2022): 2200761.Lee, Jun-Ho, et al. Nanomaterials 12.17 (2022): 3029. |
|----------------|--|

정보 탐색 능력과 끈기

석사 시절, 연구에 적용할 논문을 찾기 위해 많은 저널을 탐색하는 능력이 있으며, 관련 컨퍼런스에 참여해 적극적으로 정보를 수집했습니다.

커뮤니케이션

업무를 협업하기 위해서는 전문적인 지식배경이 비슷해야 한다는 것을 알게 됐습니다. 부족한 부분이 있으면 따로 공부를 하고 알아가며 업무를 진행합니다.

배운점

[2] 프로젝트

파이토치 기반 2D 운전
자 감지 프로그램 (DMS)
- 영상 속 졸음 감지 모델

Eyes blink 이미지를 학습하여
dlib 라이브러를 통해 얼굴 랜드마크 인식 및 OpenCV 기반 영상 인식

Keyword: Pytorch/ OpenCV/dlib/
EfficientNet/Quantization

'기존 CNN 신경망 모델링 학습 후 성능/연산량 확인 후 모델 최적화'

기여도 [80%] 참고 깃허브:
https://github.com/yunseokddi/Pytorch_dev/tree/master/sleep_detect

Standard

⇒ 깃허브를 참고하여 2D 이미지 기반 졸음 감지 모델을 OpenCV로 구현

Challenge

- ⇒ 시스템을 Develop하여 3초 이상 눈이 감길 시 경고음 발생
- ⇒ 기존 CNN모델링 이외 [논문 참고](#)하여 [EfficientNet 신경망 적용](#)
- ⇒ TensorBoard를 통해 [성능 비교 및 시각화](#)
- ⇒ 학습 시 [하드웨어 가속기 사용량 비교](#) 및 최적화 시도
- ⇒ 모델 경량화를 위해 [신경망 재구성](#) 및 [Static Quantization 적용](#)해 기존 GPU 20MB 에서 3.5MB 로 최적화

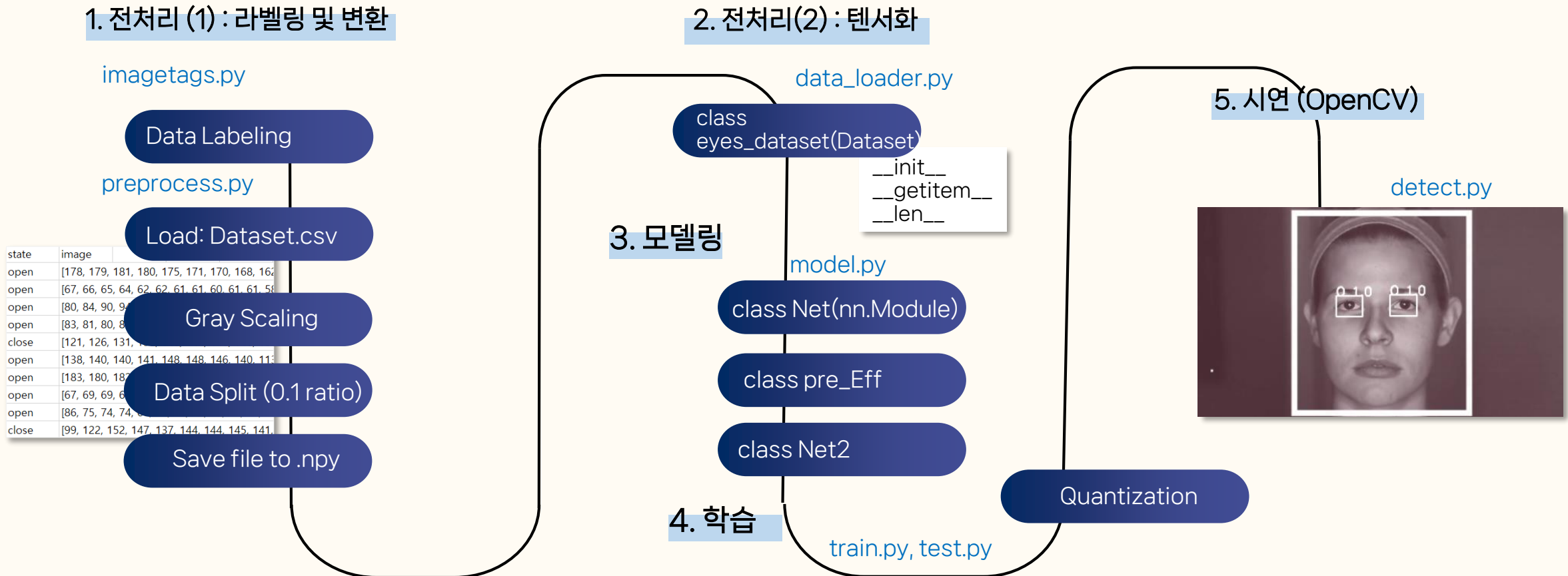
1. 프로젝트 배경

전장용 딥러닝 인공지능 기술에는 신경망 모델 구축 및 학습 뿐만 아니라 디바이스에 최적화하는 과정이 필요합니다. 기존에 만들어진 프로그램에서 느린 학습 속도를 개선하기 위해 현업에서 다루는 모델 경량화를 시도했습니다.

프로젝트

클래스를 사용해 모델을 모듈화하여 코드 유지보수를 했습니다.

2. 프로그램 프로세스



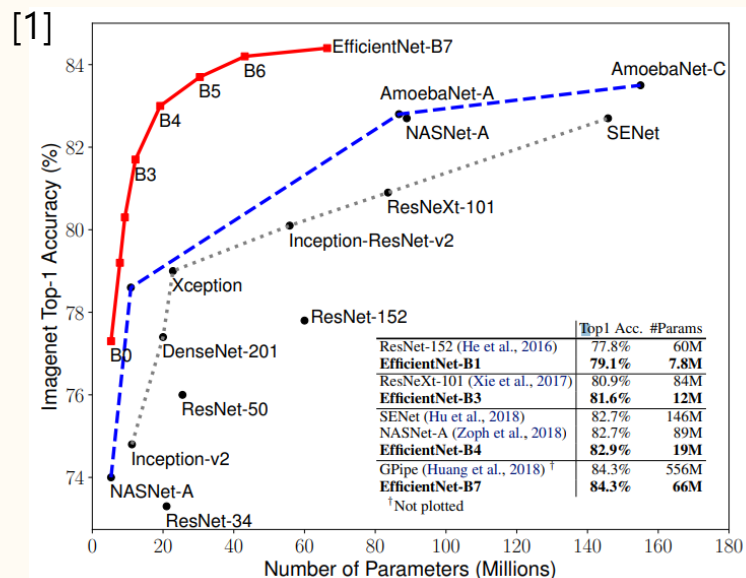
프로젝트

3. 시행착오
- (1) EfficientNet
 - (2) 성능 비교
 - (3) 양자화

(1) Efficient Net

EfficientNet 이란?

- 기존 baseline 구조에서 width, depth, resolution-scaling 을 적절히 선택해 [compound scaling](#)을 진행합니다.
- 세 scaling 파라미터의 균형을 [간단한 상수비](#)로 구해 조정합니다.
- 기초적으로 접근하기 위해 [사전학습된 EfficientNet](#) 을 활용했습니다.
- 아래 그래프와 같이 [EfficientNet-B0~B7](#)로 나뉘며, B6 이상에서는 파라미터 수에 따른 정확도가 Saturation 됩니다.



개발 환경

- OS : Window 11
- Language : Python 3.9
- Package manager : pip, conda
- GPU : RTX 3060
- CUDA : v11.7

[1] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan, Quoc V. Le

프로젝트

3. 시행착오 (1) EfficientNet
(2) 성능 비교
(3) 양자화

CNN model 1 (기준)

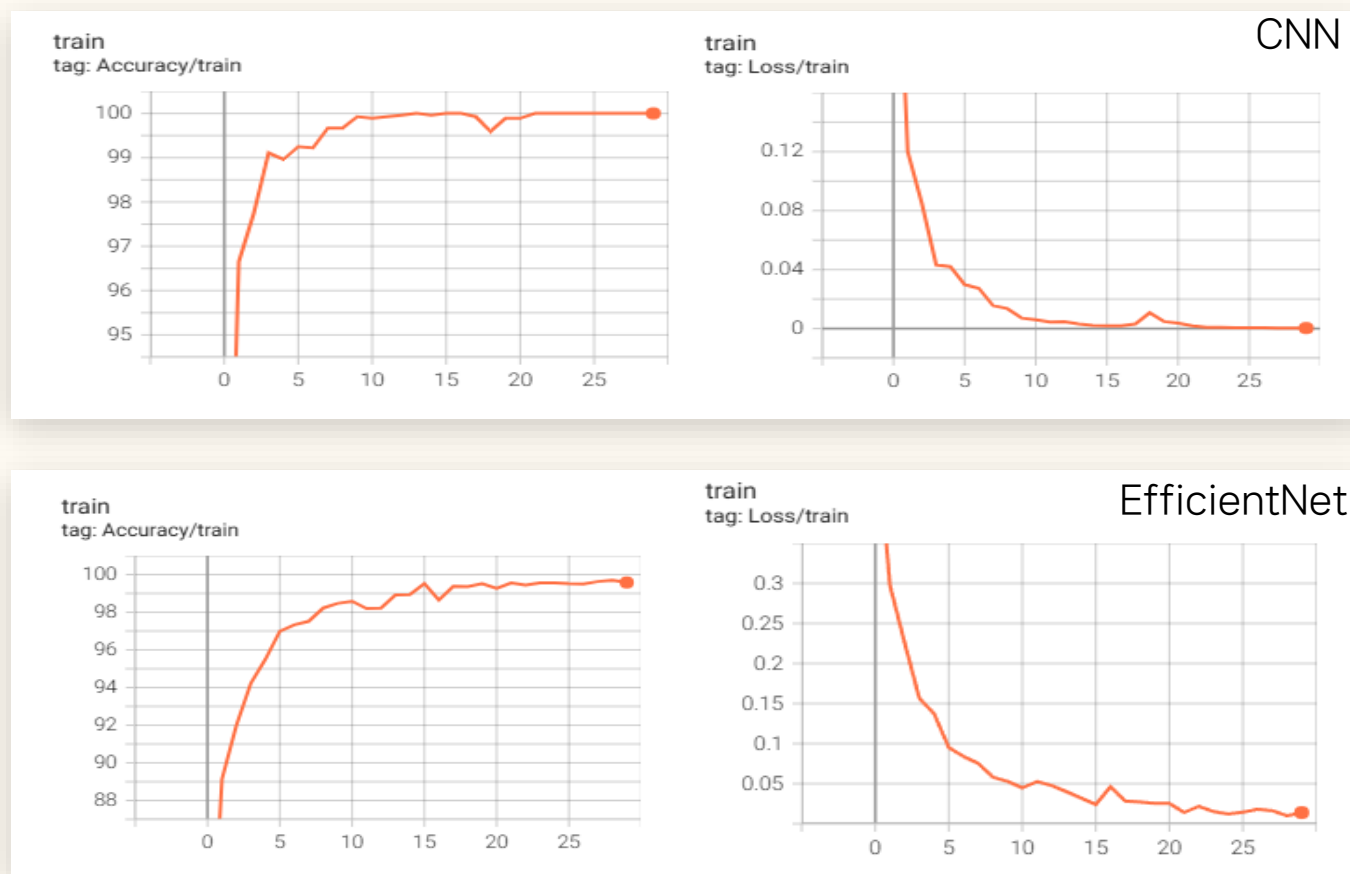
- Validation acc: 98.95
- Loss: 0.0614

Pretrained EfficientNet-B3

- Validation acc: 99.65
- Loss: 0.0274

(2) 모델 성능 비교

일반적인 신경망보다 알고리즘이 효율적인 EfficientNet 신경망을 사용해보았으나, 기존에 사용된 CNN 모델이 눈 깜박임 하나의 기능만을 위해 설계된 **간단한 모델**이기 때문에 Accuracy 및 Loss 에서 큰 차이를 보이지 않았습니다.



프로젝트

3. 시행착오
- (1) EfficientNet
 - (2) 성능 비교
 - (3) 양자화

CNN model 1 (기존)

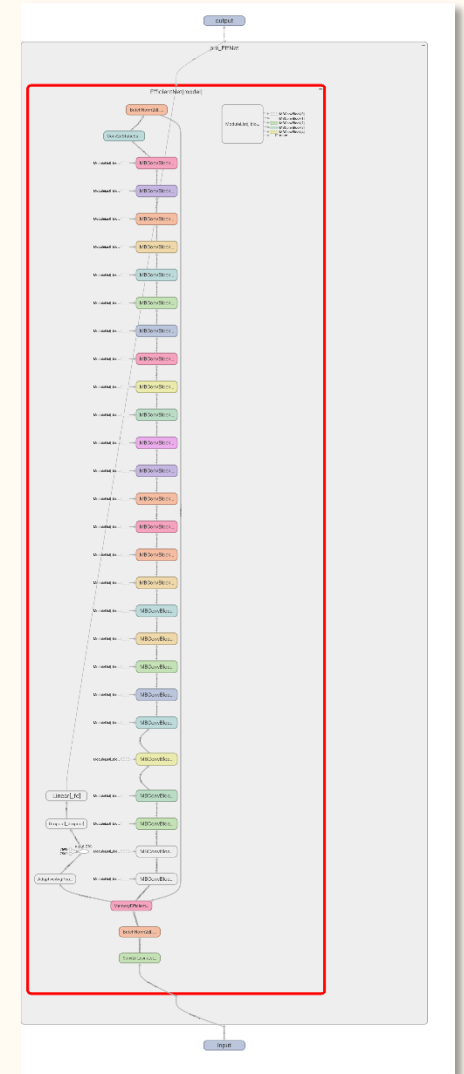
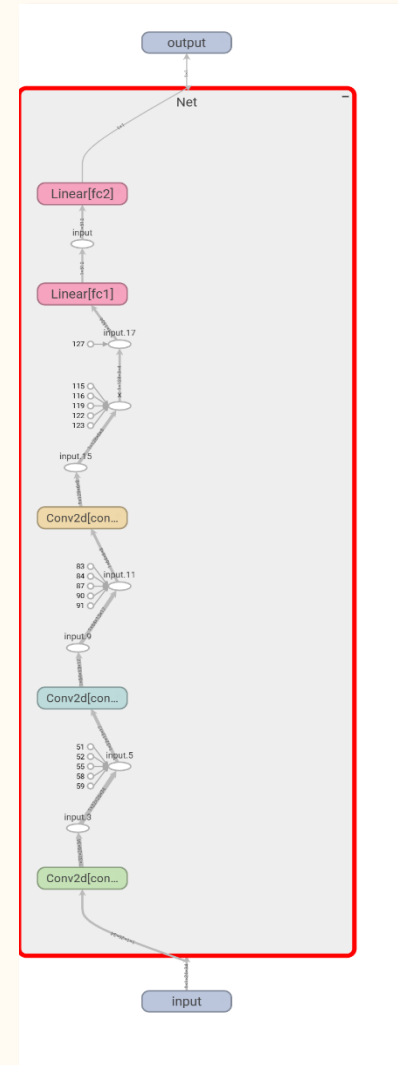
- Total GPU memory used during training: **20 MB**

Pretrained EfficientNet-B3

- Total GPU memory used during training: **135 MB**

(2) 모델 성능 비교 - 모델에 따른 메모리 연산량 체크

- 기존 모델링이 매우 간단하여 EfficientNet보다 낮은 연산량을 가진 것으로 판단했습니다.
- EfficientNet-bx 시리즈 중 b3을 사용하여 성능이 높은 대신, 메모리 사용량이 비교적 많았음을 알 수 있었습니다.



프로젝트

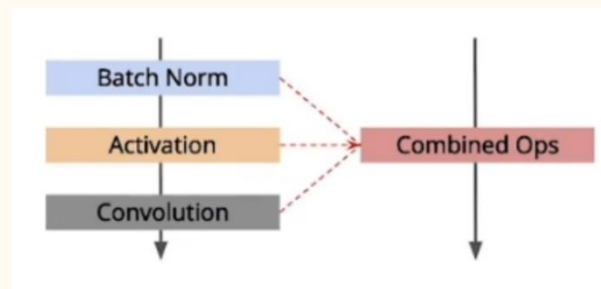
3. 시행착오
- (1) EfficientNet
 - (2) 성능 비교
 - (3) 양자화

(3) 양자화를 통해 모델 경량화 시도

신경망 재구성 후 양자화

- 부동소수점을 정수형으로 변환하여 양자화를 시도했으나, 사전학습된 Efficient model에서는 양자화가 불가능한 신경망 구조로 되어있음을 확인했습니다.
- 직접 신경망 구조를 만들어 양자화가 지원되는 레이어로 구성해 모델을 경량화했습니다.

1. CNN 모델에 배치정규화 추가 (활성화값 정규화)
2. 입력데이터 양자화 (float to int) / 출력데이터 비양자화
3. 각 Conv + BatchNorm + ReLU 층에 대한 퓨징



퓨징이란?

양자화를 개별적 레이어로 진행하면 메모리의 접근이 비효율적이기 때문에 퓨징(Fusing)을 적용했습니다. 퓨징을 함으로써 반복되는 레이어 연산을 묶고 양자화 횟수를 줄였습니다.

```
1 torch.quantization.fuse_modules(model, [['conv1', 'bn1', 'relu1']], inplace=True)
2 torch.quantization.fuse_modules(model, [['conv2', 'bn2', 'relu2']], inplace=True)
3 torch.quantization.fuse_modules(model, [['conv3', 'bn3', 'relu3']], inplace=True)
```

프로젝트

- 3. 시행착오
 - (1) EfficientNet
 - (2) 성능 비교
 - (3) 양자화

- 1. Fuse operators using
torch.quantization.fuse_modules
- 2. Quantize your model

(3) 양자화를 통해 모델 경량화 시도

Pytorch docs를 참고했으며, 모바일 CPU 전용 백엔드인 qnnpack 대신 fbgemm 을 사용했습니다.

```
import torch
from torch.utils.mobile_optimizer import optimize_for_mobile

class AnnotatedConvBnReLUModel(torch.nn.Module):
    def __init__(self):
        super(AnnotatedConvBnReLUModel, self).__init__()
        self.conv = torch.nn.Conv2d(3, 5, 3, bias=False).to(dtype=torch.float)
        self.bn = torch.nn.BatchNorm2d(5).to(dtype=torch.float)
        self.relu = torch.nn.ReLU(inplace=True)
        self.quant = torch.quantization.QuantStub()
        self.dequant = torch.quantization.DeQuantStub()

    def forward(self, x):
        x = x.contiguous(memory_format=torch.channels_last)
        x = self.quant(x)
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        x = self.dequant(x)
        return x

model = AnnotatedConvBnReLUModel()

model.qconfig = torch.quantization.get_default_qconfig('qnnpack')
torch.quantization.prepare(model, inplace=True)
# Calibrate your model
def calibrate(model, calibration_data):
    # Your calibration code here
    return
calibrate(model, [])
torch.quantization.convert(model, inplace=True)
```

프로젝트

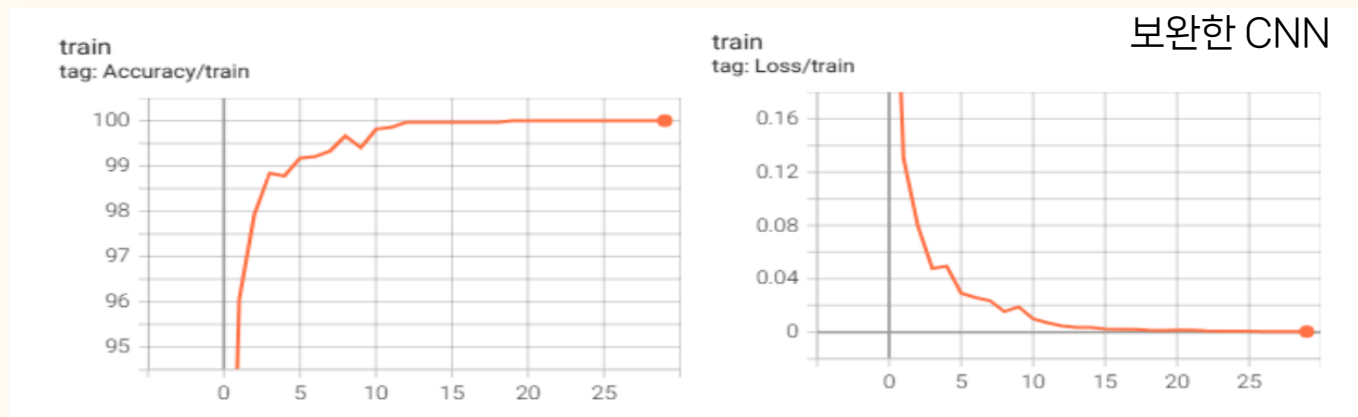
3. 시행착오
- (1) EfficientNet
 - (2) 성능 비교
 - (3) 양자화

CNN model 2

- Validation acc: 98.95
- Loss: 0.0614

(4) 모델링 재구성 및 양자화 결과

양자화 전, **신경망을 재구성**하여 기존 GPU 20MB 에서 3.5MB 로 연산량을 줄이고 학습속도를 개선하면서 성능은 Accuracy 98.95, Loss 0.06으로 높은 지표를 유지했습니다.



- 기존 CNN (train)
Total GPU memory used during training: 20 MB
- Pretrained EfficientNet -B3 (train)
Total GPU memory used during training: 135 MB



- 레이어 보완된 CNN (train)
Total GPU memory used during training: 3.5 MB
- 양자화 후 test
Total CPU memory used during validation: 1.1 MB

[3] 세부 코드 전처리 (1) : 라벨링 및 변환

데이터 설명 - Dataset.csv (2800여개의 데이터)

- 이미지 중 '눈'만 포함되어 있어 별도의 Segmentation이 불필요
- 메모리 효율성을 위해 이미지를 csv로 변환해 학습

state	image			
open	[178, 179, 181, 180, 175, 171, 170, 168, 162]			
open	[67, 66, 65, 64, 62, 62, 61, 61, 60, 61, 61, 58]			
open	[80, 84, 90, 94, 97, 100, 103, 107, 108, 112, 115]			
open	[83, 81, 80, 81, 82, 81, 79, 78, 79, 80, 81, 79]			
close	[121, 126, 131, 136, 140, 144, 146, 146, 151]			
open	[138, 140, 140, 141, 148, 148, 146, 140, 113]			
open	[183, 180, 182, 182, 179, 171, 168, 170, 171]			
open	[67, 69, 69, 69, 70, 71, 72, 72, 74, 75, 79, 76]			
open	[86, 75, 74, 74, 66, 64, 64, 58, 53, 52, 45, 46]			
close	[99, 122, 152, 147, 137, 144, 144, 145, 141, 142]			

- CSV 파일로부터 이미지 데이터와 해당 이미지의 태그(레이블)를 읽어 들이고, 이를 numpy 배열로 변환하여 반환하는 함수를 정의
- numpy 파일 : 텐서 계산 전 배열로 변환

imagetags.py

```

1 import numpy as np
2 import csv
3
4 def read_csv(path):
5     width = 34
6     height = 26
7     dims = 1
8
9     with open(path, 'r') as f:
10         # csv 파일 -> list
11         reader = csv.DictReader(f)
12         rows = list(reader)
13
14     # imgs 이미지 배열, tgs 레이블 배열 객체
15     imgs = np.empty((len(list(rows)), height, width, dims), dtype=np.uint8)
16     tgs = np.empty((len(list(rows)), 1))
17
18     for row, i in zip(rows, range(len(rows))):
19         # 리스트 행을 배열에 저장
20         img = row['image']
21         img = img.strip('[').strip(']').split(', ')
22         im = np.array(img, dtype=np.uint8)
23         im = im.reshape((height, width))
24         im = np.expand_dims(im, axis=2)
25         imgs[i] = im
26
27     # 기존 레이블 open, close -> 1, 0
28     tag = row['state']
29     if tag == 'open':
30         tgs[i] = 1
31     else:
32         tgs[i] = 0
33
34     # dataset shuffle
35     index = np.random.permutation(imgs.shape[0])
36     imgs = imgs[index]
37     tgs = tgs[index]
38
39     return imgs, tgs # 변환된 데이터셋 (이미지, 태그) 반환
40

```

전처리 (1): 라벨링 및 변환

preprocess.py

```
1 from imgetags import * # imgetags - read_csv
2 import matplotlib.pyplot as plt
3 import os, glob, cv2, random
4 import seaborn as sns
5 import pandas as pd
6 import numpy as np
7
8 base_path = 'C:/ITWILL/Video_Detection/detection/Pytorch/dataset/'
9
10 X, y = read_csv(os.path.join(base_path, 'dataset.csv'))
11
12 print(X.shape, y.shape)
13
14 # 사진 관점 왼쪽 눈
15 plt.figure(figsize=(12, 10))
16 for i in range(50):
17     plt.subplot(10, 5, i+1)
18     plt.axis('off')
19     plt.imshow(X[i].reshape((26, 34)), cmap='gray')
20
21 sns.distplot(y, kde=False)
22
23
24 # 전처리
25 n_total = len(X)
26 X_result = np.empty((n_total, 26, 34, 1))
27
28 for i, x in enumerate(X):
29     img = x.reshape((26, 34, 1)) # 1 channel (gray scaling)
30
31     X_result[i] = img
32
```

```
33 from sklearn.model_selection import train_test_split
34
35 x_train, x_val, y_train, y_val = train_test_split(X_result, y, test_size=0.1)
36
37 print(x_train.shape, y_train.shape)
38 print(x_val.shape, y_val.shape)
39 # array file 확장자 npy 사용
40 np.save(base_path + 'x_train.npy', x_train)
41 np.save(base_path + 'y_train.npy', y_train)
42 np.save(base_path + 'x_val.npy', x_val)
43 np.save(base_path + 'y_val.npy', y_val)
```

- Imgetags 모듈 사용
- Dataset split (validation 10%)

전처리 (2) : 텐서화

data_loader.py

```
1  from torch.utils.data import Dataset
2  import torch
3
4  # data_loader
5
6  class eyes_dataset(Dataset): # eyes_dataset 클래스 정의
7      def __init__(self, x_file_paths, y_file_path, transform=None):
8          self.x_files = x_file_paths
9          self.y_files = y_file_path
10         self.transform = transform # 이미지에 적용할 변환(transform)
11
12     def __getitem__(self, idx):
13         x = self.x_files[idx] # idx 위치에 있는 데이터 항목을 반환
14         x = torch.from_numpy(x).float() # 데이터 항목 텐서화 (numpy -> Pytorch Tensor)
15         y = self.y_files[idx] # idx 위치에 있는 레이블을 반환
16         y = torch.from_numpy(y).float() # 레이블 항목 텐서화 (numpy -> Pytorch Tensor)
17
18         return x, y
19
20     def __len__(self):
21         return len(self.x_files)
```

- 학습을 위해 Pytorch 텐서로 변환 (함수 정의)
- Eye_detection() 사용자 정의 함수로 dataset 전처리 후 데이터를 Pytorch의 DataLoader() 로 데이터셋 -> 모델 전달을 위한 준비 (학습을 위한 배치 자동 생성)

모델링 (1. CNN)

model.py

- 파이토치의 torch.nn 모듈로 신경망 설정

```

1 import torch.nn as nn
2 import torch.nn.functional as F
3 from torchsummary import summary
4
5 # model 1
6 class Net(nn.Module):
7     def __init__(self):
8         super(Net, self).__init__()
9
10        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
11        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
12        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
13        self.fc1 = nn.Linear(1536, 512)
14        self.fc2 = nn.Linear(512, 1)
15
16    def forward(self, x):
17        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
18        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
19        x = F.max_pool2d(F.relu(self.conv3(x)), 2)
20        x = x.reshape(-1, 1536)
21        x = F.relu(self.fc1(x))
22        x = self.fc2(x)
23
24    return x
25
26 model = Net().to('cuda') # NVIDIA GPU cuda toolkit
27 summary(model, (1,26,34))

```

Torchsummary >>

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 26, 34]	320
Conv2d-2	[-1, 64, 13, 17]	18,496
Conv2d-3	[-1, 128, 6, 8]	73,856
Linear-4	[-1, 512]	786,944
Linear-5	[-1, 1]	513

Total params: 880,129
 Trainable params: 880,129
 Non-trainable params: 0

Input size (MB): 0.00
 Forward/backward pass size (MB): 0.37
 Params size (MB): 3.36
 Estimated Total Size (MB): 3.74

모델링 (2. EfficientNet)

model.py

- 파이토치의 torch.nn 모듈로 신경망 설정

```

1 from efficientnet_pytorch import EfficientNet
2
3 class pre_EffNet(nn.Module):
4     def __init__(self, num_classes):
5         super(pre_EffNet, self).__init__()
6         # EfficientNetB3 불러오기
7         self.model = EfficientNet.from_pretrained('efficientnet-b3')
8
9         # 첫 번째 합성곱층의 입력 채널 수를 1로 변경
10        self.model._conv_stem = nn.Conv2d(
11            in_channels=1, # 1채널 입력
12            out_channels=self.model._conv_stem.out_channels,
13            kernel_size=3,
14            stride=2,
15            padding=1,
16            bias=False
17        )
18
19
20        self.model._fc = nn.Linear(self.model._fc.in_features, num_classes)
21
22    def forward(self, x):
23        return self.model(x)
24
25
26 #model = Net().to('cuda')
27 model = pre_EffNet(num_classes=1).to('cuda') # 마지막 fully connected layer의 출력 채널 수
28 summary(model, (1,26,34))

```

모델링 (3. Net2) - 배치정규화 및 양자화 추가

```

1 class Net2(nn.Module):
2     def __init__(self):
3         super(Net2, self).__init__()
4         self.quant = torch.quantization.QuantStub() # 양자화 입력
5         self.dequant = torch.quantization.DeQuantStub() # 양자화 해제 출력
6
7         self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
8         self.bn1 = nn.BatchNorm2d(32)
9         self.relu1 = nn.ReLU(inplace=True)
10
11        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
12        self.bn2 = nn.BatchNorm2d(64)
13        self.relu2 = nn.ReLU(inplace=True)
14
15        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
16        self.bn3 = nn.BatchNorm2d(128)
17        self.relu3 = nn.ReLU(inplace=True)
18
19        self.fc1 = nn.Linear(128 * 4 * 3, 512)
20        self.fc2 = nn.Linear(512, 1)
21
22    def forward(self, x):
23        x = self.quant(x) # Quantize the input
24
25        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
26        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
27        x = F.max_pool2d(F.relu(self.conv3(x)), 2)
28        x = x.reshape(-1, 128 * 4 * 3) # Flatten
29        x = F.relu(self.fc1(x))
30        x = self.fc2(x)
31
32        x = self.dequant(x) # Dequantize the output
33
34    return x

```

모델 학습 (Train)

train.py

- 배치 단위로 학습을 수행해 GPU 메모리를 효율적으로 사용

```

1 # tensorboard --logdir=C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/logs
2 import numpy as np
3 import torch
4 import torch.nn as nn # 신경망 모듈 제공
5 from torchvision.transforms import transforms
6 from torch.utils.data import DataLoader # for data loading
7 from data_loader import eyes_dataset
8 from model import Net
9 import torch.optim as optim # optimizer
10 import torchvision
11
12 # 모델링 파라미터 저장
13 PATH = 'C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/weights/'
14
15 # 데이터 로드
16 path1 = r"C:/ITWILL/Video_Detection/detection/Pytorch/dataset/"
17 x_train = np.load(path1 + 'x_train.npy').astype(np.float32) # (2586, 26, 34, 1)
18 y_train = np.load(path1 + 'y_train.npy').astype(np.float32) # (2586, 1)
19 x_train.shape
20
21
22 train_transform = transforms.Compose([
23     transforms.ToTensor(),
24     transforms.RandomRotation(10), # 학습률을 위한 변환
25     transforms.RandomHorizontalFlip(), #모델 일반화 능력 향상시키기 위한
26 ])
27
28
29 train_dataset = eyes_dataset(x_train, y_train, transform=train_transform)

```

```

33 # accuracy 함수
34 def accuracy(y_pred, y_test):
35     y_pred_tag = torch.round(torch.sigmoid(y_pred))
36
37     correct_results_sum = (y_pred_tag == y_test).sum().float()
38     acc = correct_results_sum / y_test.shape[0]
39     acc = torch.round(acc * 100)
40
41     return acc
42
43
44 #데이터셋을 배치로 나누고, 데이터를 순차적으로 로드
45 train_dataloader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4)
46 ...
47 배치 크기 32
48 전체 데이터셋 2586개의 샘플
49 총 배치 수 약 81(2586 / 32)
50 ...
51
52 model = Net()
53 model.to('cuda')
54
55 criterion = nn.BCEWithLogitsLoss() # 손실 함수
56 optimizer = optim.Adam(model.parameters(), lr=0.0001) # optimizer
57
58 epochs = 30
59
60 # 텐서보드로 로그 기록
61 from torch.utils.tensorboard import SummaryWriter
62 writer = SummaryWriter('C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/logs')

```

모델 학습 (Train)

train.py

- 배치 단위로 학습을 수행해 GPU 메모리를 효율적으로 사용

```

1 # 텐서보드로 로그 기록
2 from torch.utils.tensorboard import SummaryWriter
3 # from tensorboardX import SummaryWriter
4 # ★★★★★★★★★★
5 writer = SummaryWriter('C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/logs/pre_Eff/train')
6
7 # dummy_input 생성 및 add_graph 로그 기록
8 dummy_input = torch.randn(1, 1, 26, 34).to('cuda') # 입력 크기와 일치해야 함
9 writer.add_graph(model, dummy_input)
10
11 # 학습 전 GPU 메모리 사용량 측정
12 start_memory = torch.cuda.memory_allocated()
13
14
15 for epoch in range(epochs):
16     running_loss = 0.0
17     running_acc = 0.0
18
19     model.train() # 학습
20
21     for i, data in enumerate(train_dataloader, 0):
22         input_1, labels = data[0].to('cuda'), data[1].to('cuda')
23
24         input = input_1.transpose(1, 3).transpose(2, 3)
25
26         optimizer.zero_grad() # epoch 마다 gradient 초기화
27
28         outputs = model(input)
29
30         loss = criterion(outputs, labels)
31         loss.backward()
32         optimizer.step() # weight 업데이트
33
34         running_loss += loss.item()
35         running_acc += accuracy(outputs, labels)

```

```

36
37 # 에폭 끝난 후, 평균 손실과 정확도 계산
38 avg_loss = running_loss / len(train_dataloader) # 손실/전체 배치 수
39 avg_acc = running_acc / len(train_dataloader)
40 # 성능 로그 기록
41 writer.add_scalar('Loss/train', avg_loss, epoch)
42 writer.add_scalar('Accuracy/train', avg_acc, epoch)
43
44 print('epoch: [%d/%d] train_loss: %.5f train_acc: %.5f' % (
45     epoch + 1, epochs, avg_loss, avg_acc))
46
47 # 학습 후 GPU 메모리 사용량 측정
48 end_memory = torch.cuda.memory_allocated()
49
50 print("learning finish")
51 # 전체 학습에 사용된 메모리 출력
52 print(f'Total memory used during training: {end_memory - start_memory} bytes')
53 # ★★★★★★★★★★
54 torch.save(model.state_dict(), PATH+'cnn_train.pth')
55 torch.save(model.state_dict(), PATH+'pre_Eff_train.pth')
56
57 # SummaryWriter 닫기
58 writer.close()

```

파이토치의 훈련 모드 (model.train())

- 드롭아웃 및 배치 정규화의 훈련 기능 활성화

평가 모드 (model.eval())

- 드롭아웃이 비활성화되고, 배치 정규화가 전체 데이터셋의 통계로 정규화

모델 학습 (Test)

test.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import torch
4 import torch.nn as nn
5 from torchvision.transforms import transforms
6 from torch.utils.data import DataLoader
7 from data_loader import eyes_dataset # ★외부 모듈
8 from model import Net
9 import torch.optim as optim
10
11
12
13 # accuracy 함수
14 def accuracy(y_pred, y_test):
15     y_pred_tag = torch.round(torch.sigmoid(y_pred))
16
17     correct_results_sum = (y_pred_tag == y_test).sum().float()
18     acc = correct_results_sum / y_test.shape[0]
19     acc = torch.round(acc * 100)
20
21     return acc
22
23 PATH = 'C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/weights/trained.pth'
24 path1 = r'C:/ITWILL/Video_Detection/detection/Pytorch/dataset/'
25 x_test = np.load(path1+ 'x_val.npy').astype(np.float32) # (288, 26, 34, 1)
26 y_test = np.load(path1+ 'y_val.npy').astype(np.float32) # (288, 1)
27
28 test_transform = transforms.Compose([
29     transforms.ToTensor() ])
30

```

```

33 test_dataloader = DataLoader(test_dataset, batch_size=1, shuffle=False, num_workers=4)
34
35 model = Net()
36 model.to('cuda')
37 model.load_state_dict(torch.load(PATH)) # train된 모델의 가중치 로드
38 model.eval() # 평가 모드
39
40 count = 0 # 배치 수 count
41
42 with torch.no_grad():
43     total_acc = 0.0
44     acc = 0.0
45     for i, test_data in enumerate(test_dataloader, 0):
46         data, labels = test_data[0].to('cuda'), test_data[1].to('cuda')
47
48         data = data.transpose(1, 3).transpose(2, 3)
49
50         outputs = model(data)
51
52         acc = accuracy(outputs, labels)
53         total_acc += acc
54
55         count = i
56
57     print('average acc: %.5f' % (total_acc/count), '%')
58
59 print('test finish!')

```

양자화 시도

Quantized_test.py

```

1  """
2  Quantization of cnn
3  for test
4  정적 양자화- 이미 학습된 모델을 양자화한다.
5  """
6  import numpy as np
7  import matplotlib.pyplot as plt
8  import torch
9  import torch.nn as nn
10 from torchvision.transforms import transforms
11 from torch.utils.data import DataLoader
12 from data_loader import eyes_dataset # 외부 모듈
13 from model import Net2
14 import torch.optim as optim
15 import torch.quantization as quant
16
17 # accuracy 함수
18 def accuracy(y_pred, y_test):
19     y_pred_tag = torch.round(torch.sigmoid(y_pred))
20
21     correct_results_sum = (y_pred_tag == y_test).sum().float()
22     acc = correct_results_sum / y_test.shape[0]
23     acc = torch.round(acc * 100)
24
25     return acc
26
27 path1 = r'C:/ITWILL/Video_Detection/detection/Pytorch/dataset/'
28 x_test = np.load(path1+ 'x_val.npy').astype(np.float32) # (288, 26, 34, 1)
29 y_test = np.load(path1+ 'y_val.npy').astype(np.float32) # (288, 1)

```

```

32 # Adjust the dimensions of x_test
33 x_test = np.transpose(x_test, (0, 3, 1, 2)) # Change from (N, H, W, C) to (N, C, H, W)
34 # input 텐서의 차원 순서를 (배치 크기, 채널, 높이, 너비)로
35
36 test_transform = transforms.Compose([
37     transforms.ToTensor() ])
38
39 test_dataset = eyes_dataset(x_test, y_test, transform=test_transform)
40
41 test_dataloader = DataLoader(test_dataset, batch_size=8, shuffle=False, num_workers=4) # 배치 8로 수정
42
43 model = Net2()
44
45 # 손실 함수 정의
46 criterion = nn.BCEWithLogitsLoss()
47 count = 0 # 배치 수 count
48
49 # 양자화 준비 (pre_Eff.pth으로부터 )
50 # 1. 모델을 CPU로 이동
51
52 # cnn_train, pre_Eff_train
53 state_dict = torch.load('C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/weights/cnn2_train.pth')
54 model.load_state_dict(state_dict, strict=False) # strict=False를 사용하여 일부 키가 누락된 경우 무시
55 model.to('cpu') # 모델을 CPU로 이동
56 model.eval() # 평가 모드로 설정
57
58 # Apply fusion for Conv + BatchNorm + ReLU
59 # Conv2d + BatchNorm2d + ReLU
60 torch.quantization.fuse_modules(model, [['conv1', 'bn1', 'relu1']], inplace=True)
61 torch.quantization.fuse_modules(model, [['conv2', 'bn2', 'relu2']], inplace=True)
62 torch.quantization.fuse_modules(model, [['conv3', 'bn3', 'relu3']], inplace=True)

```


양자화 시도

Quantized_test.py

```

64 # qconfig for CPUs
65 model.qconfig = torch.quantization.get_default_qconfig('fbgemm')
66
67 # prepare
68 model = torch.quantization.prepare(model, inplace=False)
69
70 # calibration
71 def calibrate(model, dataloader):
72     model.eval()
73     with torch.no_grad():
74         for inputs, _ in dataloader:
75             inputs = inputs.to('cpu')
76             model(inputs)
77
78 # calibrate with test data
79 calibrate(model, test_dataloader)
80
81 # convert the model
82 model = torch.quantization.convert(model, inplace=False)
83
84
85 # TorchScript 모델로 저장
86 torchscript_model = torch.jit.script(model)
87 quantized_model_path = 'C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/v
88 torch.jit.save(torchscript_model, quantized_model_path)
89
90
91 print(torch.backends.quantized.engine) # 양자화 엔진 확인 # 초기값: x86 backend
92 print(torch.__version__) # PyTorch 버전 확인 # 2.0.0

```

```

97 loaded_quantized_model = torch.jit.load(quantized_model_path)
98 # quantized model 평가
99 def evaluate(model, dataloader):
100     model.eval()
101     all_preds = []
102     all_labels = []
103
104     with torch.no_grad():
105         for inputs, labels in dataloader:
106             inputs = inputs.to('cpu') # Ensure inputs are on CPU
107             labels = labels.to('cpu') # Ensure labels are on CPU
108             outputs = model(inputs)
109             all_preds.append(outputs)
110             all_labels.append(labels)
111
112     all_preds = torch.cat(all_preds)
113     all_labels = torch.cat(all_labels)
114
115     # Compute accuracy
116     acc = accuracy(all_preds, all_labels)
117     return acc
118
119
120 # 현재 프로세스의 메모리 사용량 측정
121 import psutil
122 import os
123
124 process = psutil.Process(os.getpid())
125 # 학습 전 CPU 메모리 사용량 측정
126 start_memory = process.memory_info().rss # 메모리 사용량 (바이트 단위)
127
128 test_accuracy = evaluate(loaded_quantized_model, test_dataloader)
129
130 # 학습 후 메모리 사용량 측정
131 end_memory = process.memory_info().rss # 메모리 사용량 (바이트 단위)
132 print(f'Test Accuracy: {test_accuracy:.2f}%')
133 print("--Validation finish--")
134 # 전체 학습에 사용된 메모리 출력
135 print(f'(Quantized) Total memory used during validation: {end_memory - start_memory} bytes')
136

```


Demonstration

detect.py

```

1  import cv2
2  import dlib
3  import numpy as np
4  from model import Net
5  import torch
6  from imutils import face_utils
7
8  import os
9  import time
10 import winsound
11 import threading # winsound를 프로그램과 비동기식으로 실행하기 위함
12
13
14 IMG_SIZE = (34,26)
15 PATH = 'C:/ITWILL/Video_Detection/detection/Pytorch/pytorch_code/weights/cnn_tra
16 ALERT_FILE = 'C:/ITWILL/Video_Detection/detection/Pytorch/message'
17 ALERT_SOUND = 'C:/ITWILL/Video_Detection/detection/Pytorch/sound/notify.wav'
18
19
20 detector = dlib.get_frontal_face_detector() # 얼굴을 검출하는 dlib의 얼굴 검출기.
21 predictor = dlib.shape_predictor('C:/ITWILL/Video_Detection/detection/shape_pred:
22 # 얼굴의 랜드마크를 예측하는 dlib의 랜드마크 예측기
23 model = Net()
24 model.load_state_dict(torch.load(PATH))
25 model.eval()
26
27
28 def crop_eye(img, eye_points):
29     x1, y1 = np.amin(eye_points, axis=0)
30     x2, y2 = np.amax(eye_points, axis=0)
31     cx, cy = (x1 + x2) / 2, (y1 + y2) / 2
32
33     w = (x2 - x1) * 1.2
34     h = w * IMG_SIZE[1] / IMG_SIZE[0]
35
36     margin_x, margin_y = w / 2, h / 2
37
38     min_x, min_y = int(cx - margin_x), int(cy - margin_y)
39     max_x, max_y = int(cx + margin_x), int(cy + margin_y)
40
41     eye_rect = np.rint([min_x, min_y, max_x, max_y]).astype(int)
42
43     eye_img = gray[eye_rect[1]:eye_rect[3], eye_rect[0]:eye_rect[2]]
44
45     return eye_img, eye_rect
46
47
48 def predict(pred):
49     pred = pred.transpose(1, 3).transpose(2, 3)
50     outputs = model(pred)
51     pred_tag = torch.round(torch.sigmoid(outputs))
52
53     return pred_tag

```

- Dlib 모듈의 shape_predictor_68_face_landmarks.dat 로 얼굴 랜드마크 검출

Demonstration detect.py

(1) 3초 이상 눈을 감는 경우 경고음 발생 (졸음 판정)

- 소리재생을 위한 Winsound 모듈
- 음원파일이 프레임과 비동기적으로 작동되도록 Threading 모듈 사용 (여러 thread 병렬처리)

(2) 영상에서 눈이 감지되지 않을 때 Error 메시지 로그 저장

- save_alert_to_file() 함수를 정의

```

55 # 프레임에서 눈이 감지 되었는지 확인 / 로그파일에 메세지 기록
56 def save_alert_to_file(message, folder_path):
57     if not os.path.exists(folder_path):
58         os.makedirs(folder_path)
59     with open(os.path.join(folder_path, 'alert.txt'), 'a') as file:
60         file.write(message + '\n')
61
62 # 3초 이상 눈 감는 경우 경고음 실행
63 def play_alert_sound():
64     winsound.PlaySound(ALERT_SOUND, winsound.SND_FILENAME)

```

```

68 n_count = 0
69 alert_playing = False # 알람이 울리는 상태를 추적하기 위한 변수
70
71
72 # 비디오 캡처 객체를 저장된 비디오 파일로 초기화
73 video_path = 'C:/ITWILL/Video_Detection/detection/Pytorch/videos/3.mp4'
74 cap = cv2.VideoCapture(video_path)
75
76 # GIF 저장을 위한 프레임 리스트
77 frames = []
78
79 while cap.isOpened():
80     ret, img_ori = cap.read()
81     if not ret:
82         break
83
84     img_ori = cv2.resize(img_ori, dsize=(0, 0), fx=0.5, fy=0.5)
85
86     img = img_ori.copy()
87     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
88
89     faces = detector(gray) # dlib의 얼굴검출기로 grayscale 이미지에서 얼굴 검출
90
91     eyes_detected = False
92
93     for face in faces: # 얼굴 랜드마크 검출 및 눈 영역 추출
94         shapes = predictor(gray, face)
95         shapes = face_utils.shape_to_np(shapes)
96
97         # 눈의 랜드마크 좌표를 기반으로 눈 영역 crop
98         eye_img_l, eye_rect_l = crop_eye(gray, eye_points=shapes[36:42])
99         eye_img_r, eye_rect_r = crop_eye(gray, eye_points=shapes[42:48])
100
101         eye_img_l = cv2.resize(eye_img_l, dsize=IMG_SIZE)
102         eye_img_r = cv2.resize(eye_img_r, dsize=IMG_SIZE)
103         eye_img_r = cv2.flip(eye_img_r, flipCode=1)
104

```

Demonstration detect.py

```

105 # 모델 예측
106 # 눈 이미지 데이터 Pytorch 텐서로 변환 후 모델 입력
107 eye_input_l = eye_img_l.copy().reshape((1, IMG_SIZE[1], IMG_SIZE[0], 1)).astype(np.float32)
108 eye_input_r = eye_img_r.copy().reshape((1, IMG_SIZE[1], IMG_SIZE[0], 1)).astype(np.float32)
109
110 eye_input_l = torch.from_numpy(eye_input_l)
111 eye_input_r = torch.from_numpy(eye_input_r)
112
113 # 예측 결과 반환
114 pred_l = predict(eye_input_l) # 왼눈
115 pred_r = predict(eye_input_r) # 오른눈
116
117 # 예측 결과 처리 및 시각화
118 # 예측 결과를 바탕으로 눈 감김 상태를 체크하고 특정 조건을 만족하면 경고 메시지를 화면에 출력
119 if pred_l.item() == 0.0 and pred_r.item() == 0.0:
120     n_count += 1
121     eyes_detected = True
122 else:
123     n_count = 0 # 프레임 카운트 초기화
124     alert_playing = False # 눈을 떴을 때 알람 상태 초기화
125
126 if n_count > 90: # 90 프레임 (~3초)
127     if not alert_playing: # 알람이 울리지 않는 상태에서만 실행
128         cv2.putText(img, "Wake up", (120, 160), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
129
130         threading.Thread(target=play_alert_sound).start() # 알람을 비동기적으로 실행
131         alert_playing = True # 알람 상태를 활성화
132
133 # visualize
134 state_l = '0 %.1f' if pred_l > 0.1 else '- %.1f'
135 state_r = '0 %.1f' if pred_r > 0.1 else '- %.1f'
136
137 state_l = state_l % pred_l
138 state_r = state_r % pred_r

```

```

140 if n_count > 90:
141     cv2.rectangle(img, pt1=tuple(eye_rect_l[0:2]), pt2=tuple(eye_rect_l[2:4]), color=(0,0,255), thickness=2)
142     cv2.rectangle(img, pt1=tuple(eye_rect_r[0:2]), pt2=tuple(eye_rect_r[2:4]), color=(0,0,255), thickness=2)
143     cv2.putText(img, state_l, tuple(eye_rect_l[0:2]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
144     cv2.putText(img, state_r, tuple(eye_rect_r[0:2]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)
145
146 else:
147     cv2.rectangle(img, pt1=tuple(eye_rect_l[0:2]), pt2=tuple(eye_rect_l[2:4]), color=(255,255,255), thickness=2)
148     cv2.rectangle(img, pt1=tuple(eye_rect_r[0:2]), pt2=tuple(eye_rect_r[2:4]), color=(255,255,255), thickness=2)
149     cv2.putText(img, state_l, tuple(eye_rect_l[0:2]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
150     cv2.putText(img, state_r, tuple(eye_rect_r[0:2]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
151
152 #cv2.putText(img, state_l, tuple(eye_rect_l[0:2]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
153 #cv2.putText(img, state_r, tuple(eye_rect_r[0:2]), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
154
155 if not eyes_detected:
156     save_alert_to_file("눈이 감지되지 않음", ALERT_FILE)
157
158 # 프레임을 리스트에 추가
159 frames.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
160
161 cv2.imshow('result', img) # 결과 프레임을 화면에 표시
162 if cv2.waitKey(1) == ord('q'):
163     break
164
165 cap.release()
166 cv2.destroyAllWindows()
167
168 # GIF로 저장
169 import imageio
170 gif_path = 'C:/Users/minjeong/Documents/itwill/Video_Detection/detection/Pytorch/videos/output.gif'
171 with imageio.get_writer(gif_path, mode='I', duration=0.1) as writer:
172     for frame in frames:
173         writer.append_data(frame)

```

[4]

레퍼런스

- 기본 코드
- https://github.com/yunseokddi/Pytorch_dev/tree/master/sleep_detect
- 여러가지 네트워크 경량화 기법에 대한 글
- <https://velog.io/@ailab/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%AA%A8%EB%8D%B8-%EA%B2%BD%EB%9F%89%ED%99%94>
- 딥러닝 네트워크 압축 기술에 대한 개요
- <https://sotudy.tistory.com/12>
- Mobilenet 에 대한 구현 /github
- <https://deep-learning-study.tistory.com/549>
- <https://github.com/weiaicunzai/pytorch-cifar100/blob/master/models/mobilenet.py>
- 파이토치 Docs
- https://tutorials.pytorch.kr/beginner/blitz/cifar10_tutorial.html
- <https://pytorch.org/blog/introduction-to-quantization-on-pytorch/#device-and-operator-support>
- 양자화 개념 정리
- <https://velog.io/@jooh95/%EB%94%A5%EB%9F%AC%EB%8B%9D-Quantization%EC%96%91%EC%9E%90%ED%99%94-%EC%A0%95%EB%A6%AC>