

Facial-Expression Recognition Speaker

-Embedded AI Term Project-

기계IT대학 전자공학과

21611648 유준상

담당교수 : 김성호

Table of Contents

1. Introduction

2. Design details

3. Results

4. Conclusion

5. Source code

6. Reference

1. Introduction

스마트 스피커, 흔히 AI 스피커라고 불리는 제품은 출시된 이후 편리함으로 가정, 회사 곳곳에 보급되었다. 주로 사용하는 기능은 음악 재생, 뉴스 또는 일기예보 읽기 등등이다. AI 스피커의 가장 큰 장점은 사용자가 직접 손으로 조작하지 않고, 음성 인식으로 원하는 어플리케이션 실행이나 기능을 사용할 수 있다. 따라서 사용자는 하던 작업을 집중하는 것이 가능하다.

현재 세계 탑 IT회사에서는 이미 AI스피커를 판매 중이거나 판매 계획을 가지고 있다. 국내에서는 통신사와 협업을 통해 판매, SNS 서비스 운영 회사 등이 판매 중이다. 그리고, 꼭 AI 스피커가 아니더라도 대부분의 사람의 스마트폰에는 해당 기능이 있다. 애플 사의 아이폰은 'Siri', 삼성전자 사의 갤럭시 시리즈는 'Bixby'이다. 스마트폰 제조사에 따른 기본 탑재된 인공지능 비서라고 칭하는 위의 예 외에도 앞서 말한대로, 구글, 카카오톡 등 웹, SNS 서비스 회사 이외에도 최근은 음원 서비스 회사도 해당 기능을 제공하고 있다.

본인은 평소 노래를 즐겨 듣는다. 주로 듣는 장르는 외국 힙합이지만, 가끔 기분이 꿀꿀할 때는 브릿팝이 듣고 싶어질 때가 있다. 그럴 때 본인은, "시리야, 오아시스 노래 틀어줘" 라던가 "기분이 꿀꿀할 때 듣는 노래 틀어줘" 과 같은 방법으로 노래를 재생시키는 편이다. 처음에는 음성인식 기능을 사용하는 것이 익숙지 않고 낯 부끄러워 잘 사용하지 않았다. 하지만, 몇 번 사용하다 보니 너무 편해서 자주 사용하는 방법이다. 주로 사용하는 기능은 오늘 날씨와 음악 재생이다.

너무 편한 기능이지만 사용하다 보니 아쉬운 점이 몇 개 보이기 시작했다. 첫 번째는, 음성 인식 기술의 수준이다. 짧은 문장이나 사람들이 자주 사용하는 일기예보 같은 것들은 잘 인식되지만, 문장이 길어지거나 두 가지 언어를 같이 쓰면 인식이 잘 안되는 점이다. 예를 들어, "Siri야, Oasis의 <Don't look back in anger> 틀어줘" 라고 말하면, "시리야, 오아시스의 돈룩 백인 앵거 틀어줘" 등으로 인식하고 해당 시스템은 잘 모르겠다고 다시 말해달라고 한다.

두 번째는 내가 어떤 스타일의 음악을 듣고 싶을 때 직접 지정해서 말해주지 않고 음악을 재생하는 기능이다. 현재는 위치를 듣고 싶은 한 노래나, 노래 장르를 지정해주어야 재생된다. 지금도 물론 편하지만, 시스템이 내 표정을 읽고 표정에 맞는 노래를 재생해주면 더 좋을 것 같다. 예를 들면, <우울한 표정>이라면 <감성 발라드> 재생 같은 형식이다.

이 아이디어를 발전시킨다면, AI스피커의 새로운 단계로 나아갈 것이라고 생각한다. 현재 음성 인식의 한계를 뛰어넘는 AI스피커 2.0 단계라고 생각한다. 조금 더 발전해가면, 비단 음악 재생에서만 사용할 뿐만 아니라, 만약 두려움을 느끼는 표정을 감지한다면 경찰로 해당 정보를 전송하는 등의 방법도 있다.

2. Design details

1) Step-by-Step Design

1) facial expression recognition을 위한 모델 train



2) trained 모델 불러오기



3) 카메라 셋팅 및 불러오기



4) Detector 셋팅



5) Detector로 현재 프레임에서 얼굴 찾기



6) 찾은 얼굴에서 표정 인식



7) 인식한 표정에 맞는 LED 제어



8) 실시간 영상 및 인식 결과 출력



9) 반복

2) Requirements

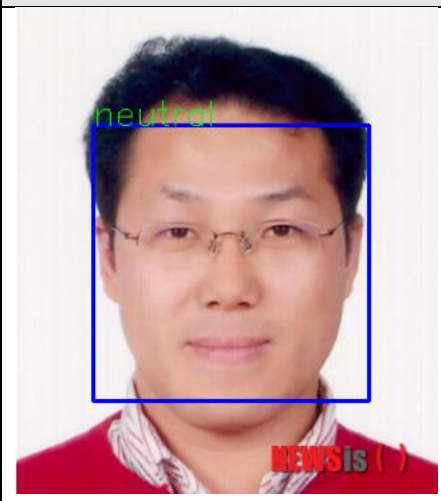
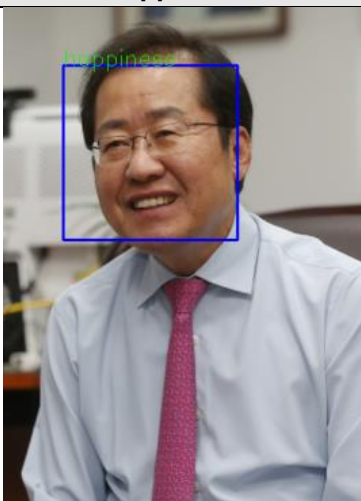

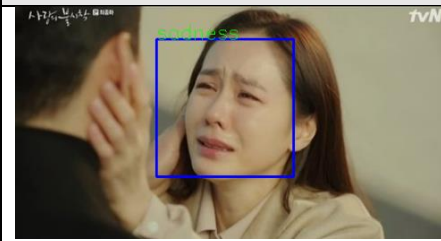

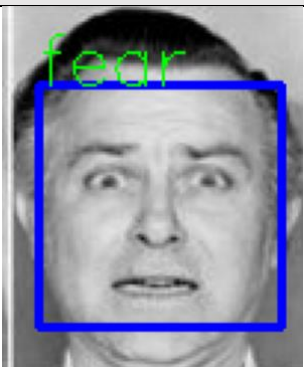
Sort	Specification
Platform	NVIDIA Jetson nano
OS	Ubuntu 18.04
Device	Input : RPi camera
	Output : monitor
	GPIO : LED 3 colors(red, yellow, green)
Library	<pre> (test13) nano@nano-desktop:~/FER_NET_RT\$ pip list Package Version ----- cycler 0.11.0 Cython 0.29.24 dataclasses 0.8 Jetson.GPIO 2.0.17 joblib 1.1.0 kiwisolver 1.3.1 matplotlib 3.3.4 numpy 1.19.5 pandas 1.1.5 Pillow 8.4.0 pip 21.3.1 pyparsing 3.0.6 python-dateutil 2.8.2 pytz 2021.3 scipy 1.5.4 setuptools 58.3.0 six 1.16.0 threadpoolctl 3.0.0 torch 1.8.0 torchvision 0.9.0a0+01dfa8e typing_extensions 4.0.0 wheel 0.37.0 </pre>

3. Results

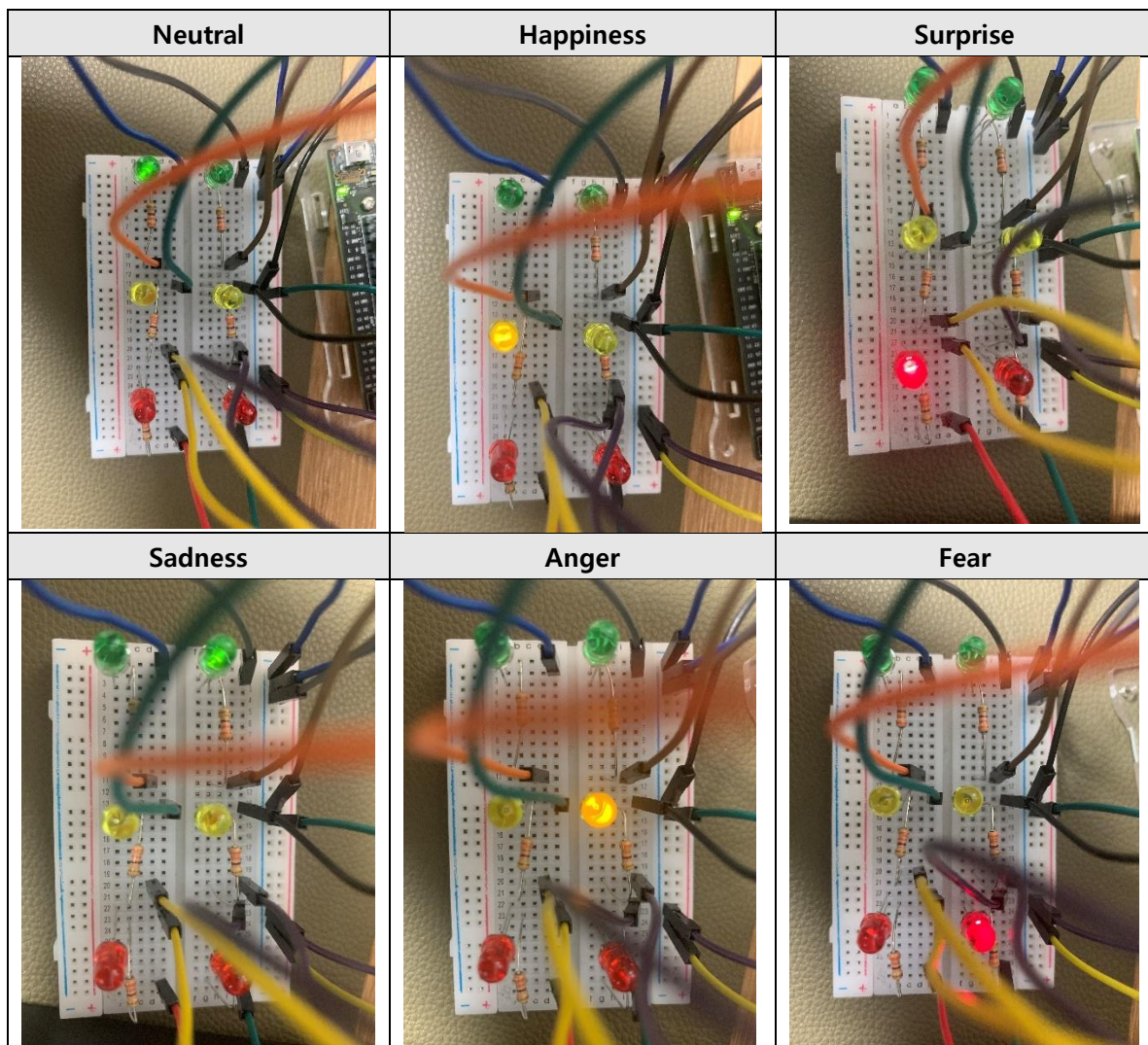
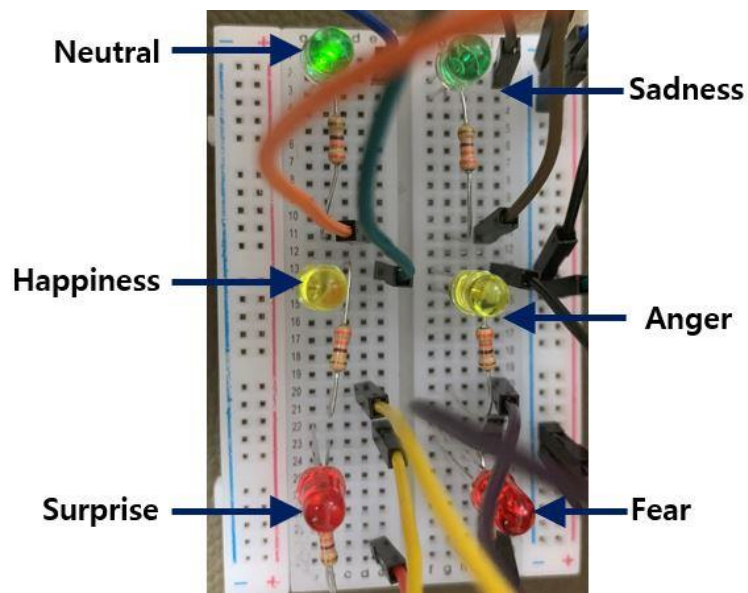
1) Image

실시간 영상에서의 구현이 목표지만, 모델 성능 및 LED 제어 연결 테스트를 위해 만들었다. 전체적인 구조는 2에서 나타낸 Real-time의 Design의 Flow와 비슷하다. 다른 점은 실시간 영상처럼 입력이 항상 같지 않으므로, 실행마다 경로를 지정해 주어야한다.

[결과]

Neutral	Happiness	Surprise
		
Sadness	Anger	Fear
		

[LED]



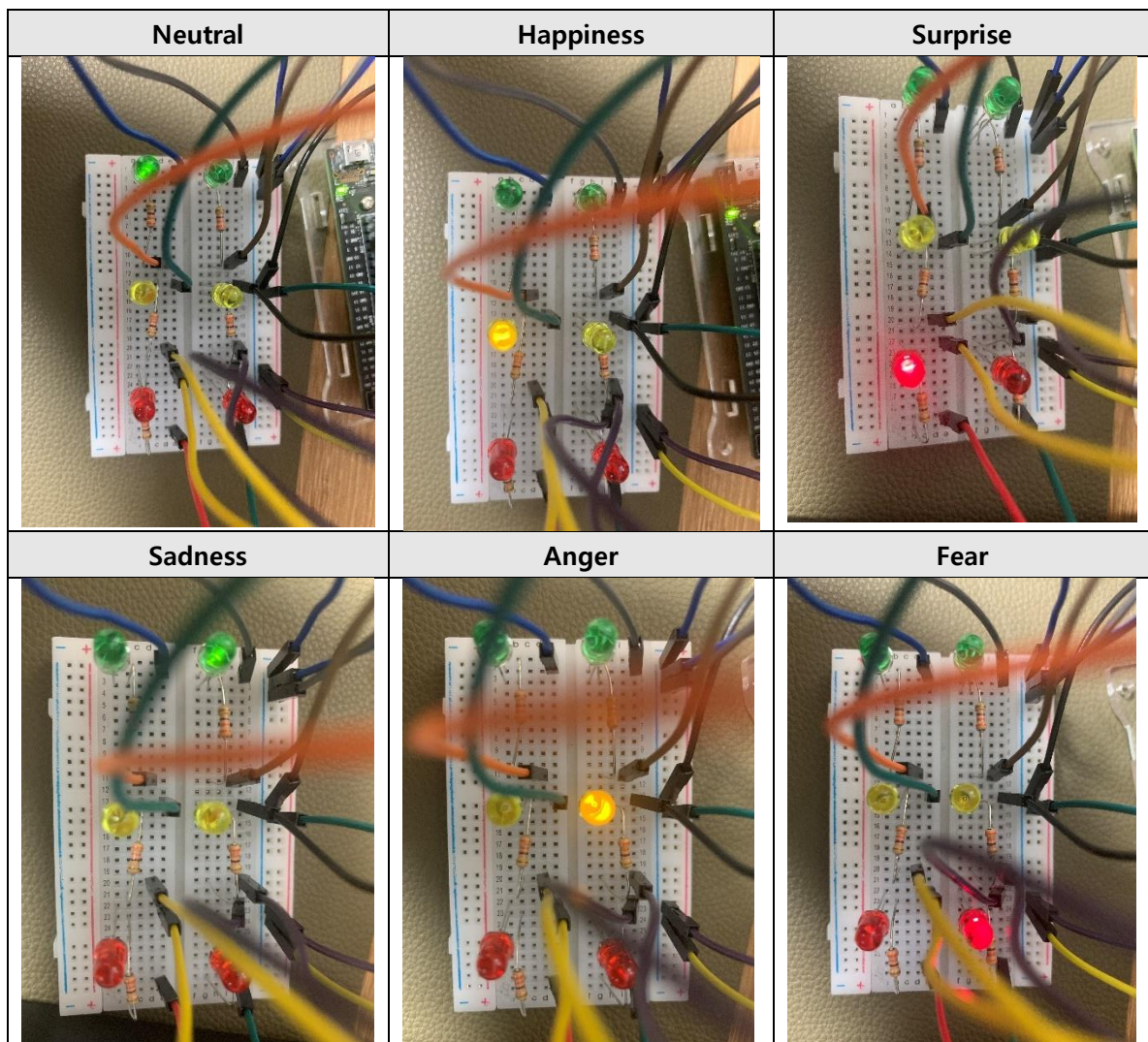
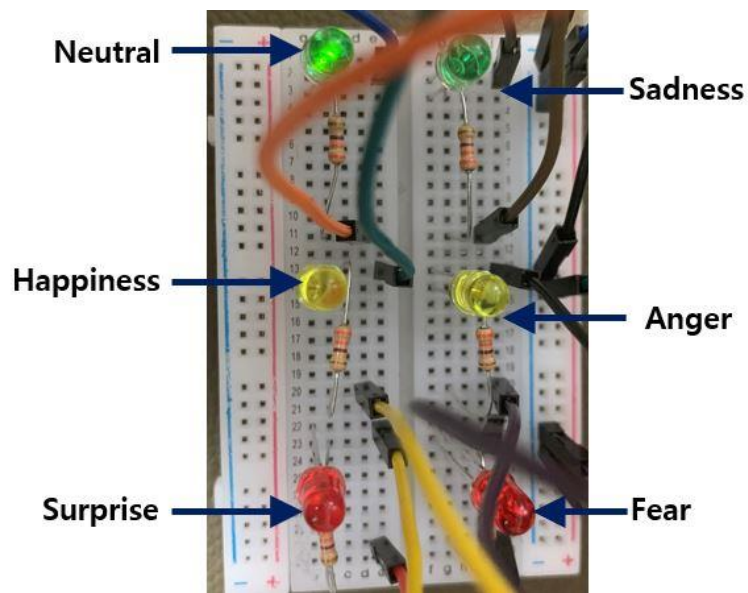
2) Real-time

1)과 달리 소스코드만 실행하면 된다. 1)에서는 입력 1개당 결과가 1개이기 때문에 LED도 하나지만, Real-time에서는 입력 카메라가 연결되어 있는 동안 여러 LED가 켜고 꺼진다. 해당 코드의 Flow는 2에서의 Design과 같다.

[결과]

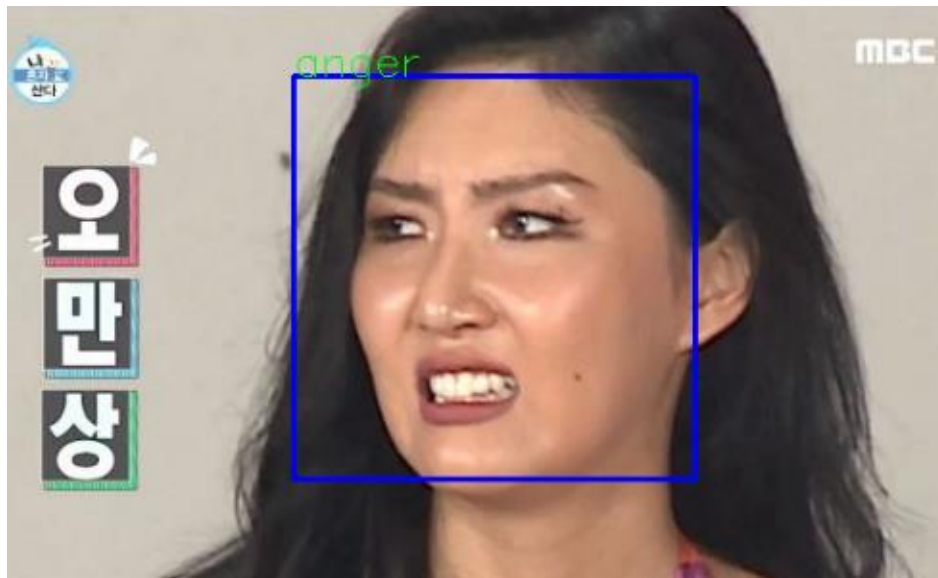


[LED]



4. Conclusion / Discussion

Project의 목표는 실시간 영상을 입력하면, 학습한 모델로 얼굴 표정을 인식하는 것이었다. 디자인하는 과정에서 비슷한 Flow인 가벼운 테스트 용으로 이미지 하나를 입력하면 해당 이미지에서 얼굴을 찾고 찾은 얼굴의 표정을 인식해서 박스 친 후 예측한 결과를 출력하는 기능을 한다. 테스트용 예시는 아래와이다.



모델이 충분히 기능을 한다고 판단하여 원래 목표인 실시간 영상으로 얼굴 표정 인식을 구현했다. 코드에 print 함수를 넣어서 터미널에서 예측 결과를 출력했지만, 원래 최종 구상안인 AI 스피커 느낌을 주기위해 예측 결과가 다른 Output으로의 연결을 새로운 목표로 정했다. 스피커가 없어서 스피커 형태로 제작은 불가능했지만, 간단한 회로 구성으로 LED 제어로 정했다.

카메라를 통해 입력된 실시간 영상을 학습된 모델이 얼굴 표정 인식하고 인식한 각 표정 결과에 맞는 LED를 On하는 Flow이다. 원 결과 예측 코드와 같이 6개의 표정(Neutral, Happiness, Surprise, Sadness, Anger, Fear)에 6개의 다른 LED 3색(red, green, yellow)을 사용하여 시각적으로 논리적으로 결과가 잘 나오고 잘 전달됨을 보였다.

이번 프로젝트에서 처음으로 NVIDIA Jetson nano를 가지고 프로젝트를 해봤다. 앞서 여러 실습을 통해 기본기를 다져서 전체적으로 프로젝트의 진행은 순조로웠다. 다만, 일반적인 시스템들과 다르게 ubuntu에 여러 라이브러리를 설치하는 것이 어려웠다. 특히 tensorflow와 관련 라이브러리들은 설치가 잘 안되어 중간에 여러 번 프로젝트를 옆었다. 그것만 제외하면 어차피 같은 Ubuntu 위에 진행하는 것이기에 순조롭게 진행하였다. 대학교 4년 마지막 프로젝트로, 직접 여러가지 시도해보고 고민하고 만들어가고 결과가 나온 성공적인 프로젝트를 한 것 같아 뜻깊고 좋았다.

5. Source code

[dataset.py] : 데이터셋 로드 / 76 lines

```
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch.utils.data as utils
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image

emotion_dict = {0: 'angry', 1: 'disgust', 2: 'fear', 3: 'happy',
                4: 'sad', 5: 'surprise', 6: 'neutral'}

def load_fer2013(path_to_fer_csv):
    data = pd.read_csv(path_to_fer_csv)
    pixels = data['pixels'].tolist()
    width, height = 48, 48
    faces = []
    for pixel_sequence in pixels:
        face = [int(pixel) for pixel in pixel_sequence.split(' ')]
        face = np.asarray(face).reshape(width, height)
        face = cv2.resize(face.astype('uint8'), (48,48))
        faces.append(face.astype('float32'))
    faces = np.asarray(faces)
    faces = np.expand_dims(faces, -1)
    emotions = data['emotion'].values
    return faces, emotions

def show_random_data(faces, emotions):
    idx = np.random.randint(len(faces))
    print(emotion_dict[emotions[idx]])
    plt.imshow(faces[idx].reshape(48,48), cmap='gray')
    plt.show()

class EmotionDataset(utils.Dataset):
    def __init__(self, X, y, transform=None):
        self.X = X
        self.y = y
        self.transform = transform
```

```

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

def __getitem__(self, index):
    x = self.X[index].reshape(48,48)
    x = Image.fromarray((x))
    if self.transform is not None:
        x = self.transform(x)
    y = self.y[index]
    return x, y

def get_dataloaders(path_to_fer_csv='', tr_batch_sz=3000, val_batch_sz=500):
    faces, emotions = load_fer2013(path_to_fer_csv)
    train_X, val_X, train_y, val_y = train_test_split(faces, emotions,
test_size=0.2,
                                                    random_state = 1,
shuffle=True)
    train_transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(30),
        transforms.ToTensor(),
        transforms.Normalize((0.507395516207, ),(0.2551289894
15, ))
    ])
    val_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.507395516207, ),(0.255128989415,
))
    ])

    train_dataset = EmotionDataset(train_X, train_y, train_transform)
    val_dataset = EmotionDataset(val_X, val_y, val_transform)

    trainloader = utils.DataLoader(train_dataset, tr_batch_sz)
    validloader = utils.DataLoader(val_dataset, val_batch_sz)

```

[model.py] : model 구현 / 61 lines

```
import torch.nn as nn
import torch

class FER_Net(nn.Module):

    def __init__(self):

        super(FER_Net, self).__init__()

        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=3)
        self.cnn2 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3)
        self.cnn3 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3)
        self.cnn4 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
        self.cnn5 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3)
        self.cnn6 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3)
        self.cnn7 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3)
        self.relu = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2, 1)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.cnn1_bn = nn.BatchNorm2d(8)
        self.cnn2_bn = nn.BatchNorm2d(16)
        self.cnn3_bn = nn.BatchNorm2d(32)
        self.cnn4_bn = nn.BatchNorm2d(64)
        self.cnn5_bn = nn.BatchNorm2d(128)
        self.cnn6_bn = nn.BatchNorm2d(256)
        self.cnn7_bn = nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 7)
        self.dropout = nn.Dropout(0.3)
        self.log_softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):

        x = self.relu(self.pool1(self.cnn1_bn(self.cnn1(x))))
        x = self.relu(self.pool1(self.cnn2_bn(self.dropout(self.cnn2(x)))))
        x = self.relu(self.pool1(self.cnn3_bn(self.cnn3(x))))
        x = self.relu(self.pool1(self.cnn4_bn(self.dropout(self.cnn4(x)))))
        x = self.relu(self.pool2(self.cnn5_bn(self.cnn5(x))))
        x = self.relu(self.pool2(self.cnn6_bn(self.dropout(self.cnn6(x)))))
        x = self.relu(self.pool2(self.cnn7_bn(self.dropout(self.cnn7(x)))))

        x = x.view(x.size(0), -1)

        x = self.relu(self.dropout(self.fc1(x)))
```

```
x = self.relu(self.dropout(self.fc2(x)))
x = self.log_softmax(self.fc3(x))

return x

def count_parameters(self):

    return sum(p.numel() for p in self.parameters() if p.requires_grad)

if __name__ == '__main__':

    bn_model = FER_Net()
    x = torch.randn(1,1,48,48)
    print('Shape of output = ',bn_model(x).shape)
    print('No of Parameters of the BatchNorm-CNN Model
    =',bn_model.count_parameters())
```


[train.py] : 모델 학습 / 87 lines

```
import torch
import torch.nn as nn
from model import *
from dataset import *
import matplotlib.pyplot as plt

def train_model(model, trainloader, validloader, epochs=100,
visualize_learning_curve=True):
    criterion = nn.NLLLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    valid_loss_min = np.Inf
    train_losses, test_losses = [], []
    for e in range(epochs):
        model.train()
        running_loss = 0
        tr_accuracy = 0
        for images, labels in trainloader:
            images = images.cuda()
            labels = labels.long().cuda()
            optimizer.zero_grad()

            log_ps = model(images)
            loss = criterion(log_ps, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            ps = torch.exp(log_ps)
            top_p, top_class = ps.topk(1, dim=1)
            equals = top_class == labels.view(*top_class.shape)
            tr_accuracy += torch.mean(equals.type(torch.FloatTensor))
        else:
            test_loss = 0
            accuracy = 0
            with torch.no_grad():
                model.eval()
                for images, labels in validloader:
                    images = images.cuda()
                    labels = labels.long().cuda()
                    log_ps = model(images)
                    test_loss += criterion(log_ps, labels)

                ps = torch.exp(log_ps)
                top_p, top_class = ps.topk(1, dim=1)
```

```

        equals = top_class == labels.view(*top_class.shape)
        accuracy += torch.mean(equals.type(torch.FloatTensor))

    train_losses.append(running_loss/len(trainloader))
    test_losses.append(test_loss/len(validloader))

    print("Epoch: {}/{}".format(e+1, epochs),
          "Training Loss: {:.3f} ".format(train_losses[-1]),
          "Training Acc: {:.3f}
".format(tr_accuracy/len(trainloader)),
          "Val Loss: {:.3f} ".format(test_losses[-1]),
          "Val Acc: {:.3f}".format(accuracy/len(validloader)))
    if test_loss/len(validloader) <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}). Saving
model ...'.format(
            valid_loss_min,
            test_loss/len(validloader))
        torch.save(model.state_dict(), 'best_model.pt')
        valid_loss_min = test_loss/len(validloader)

    if visualize_learning_curve:
        plt.plot(train_losses, 'b', label='Training Loss')
        plt.plot(test_losses, 'r', label='Validation Loss')
        plt.show()
    return model

def main():
    trainloader, validloader = get_dataloaders()
    print('Data Preprocessed and got DataLoaders...')

    model = FER_Net()
    if torch.cuda.is_available():
        model.cuda()
        print('GPU Found!!!, Moving Model to CUDA.')
    else:
        print('GPU not found!!, using model with CPU.')

    print('Starting Training loop...\n')
    model = train_model(model, trainloader, validloader, epochs=200)

if __name__ == '__main__':
    main()

```

[FER_NET_img.py] : 입력 이미지 얼굴 표정 인식 / 118 lines

```
import cv2
import torch
import torchvision.transforms as transforms
from PIL import Image
import matplotlib.pyplot as plt
import argparse
import os
from model import *
import RPi.GPIO as GPIO
import time
from multiprocessing import Process

# Load model
def load_trained_model(model_path):

    model = FER_Net()
    model.load_state_dict(torch.load(model_path, map_location = lambda
storage, loc : storage), strict = False)

    return model

# Recognition
def FER_img(img_path):

    # 1) Load model
    model = load_trained_model('./models/FER_trained_model.pt')

    expression_dict = {0: 'neutral', 1: 'happiness', 2: 'surprise', 3:
'sadness',
                        4: 'anger', 5: 'disgust', 6: 'fear'}

    val_transform = transforms.Compose([
        transforms.ToTensor()])

    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 2) Set detector
    FER_cascade =
cv2.CascadeClassifier('./models/haarcascade_frontalface_default.xml')
    face = FER_cascade.detectMultiScale(img)

    # 3) Predict input image
    for (x, y, w, h) in face:
```

```

cv2.rectangle(img, (x,y), (x+w, y+h), (255,0,0), 2)
resize_frame = cv2.resize(gray[y:y + h, x:x + w], (48, 48))
X = resize_frame/256
X = Image.fromarray((resize_frame))
X = val_transform(X).unsqueeze(0)

with torch.no_grad():
    model.eval()
    log_ps = model.cpu()(X)
    ps = torch.exp(log_ps)
    top_p, top_class = ps.topk(1, dim=1)
    pred = expression_dict[int(top_class.numpy())]
    cv2.putText(img, pred, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,
255, 0), 1)

print("Prediction : ",pred)

# 4) Plot Prediction
def plot():

    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.grid(False)
    plt.axis('off')
    plt.show()

# 5) LED Control - for predictions

# GPIO - Board mode
output_dict = {'neutral':13, 'happiness' : 7, 'surprise': 29,
'sadness':40,
                'anger': 12, 'fear': 24, 'disgust':''}

output_pin = output_dict[pred]
print("Output Pin : ",output_pin)

def led():

    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(output_pin,GPIO.OUT,initial=GPIO.HIGH)
    print("Press CTRL+C when you want hte LED to stop blinking")

    try:
        while True:
            time.sleep(0.5)
            GPIO.output(output_pin, GPIO.HIGH)
            print("LED ON")

```

```

        time.sleep(0.5)
        GPIO.output(output_pin, GPIO.LOW)
        print("LED OFF")
    finally:
        GPIO.cleanup()

# Use multiprocessing - to output concurrently(plot&led)
p1 = Process(target=plot)
p2 = Process(target=led)

p1.start()
p2.start()

p2.join()
p1.join()

GPIO.cleanup()

if __name__ == "__main__":

    # Need image path to Run
    ap = argparse.ArgumentParser()
    ap.add_argument("-p", "--path", required=True,
                    help="path of image")
    args = vars(ap.parse_args())

    if not os.path.isfile(args['path']):
        print('The image path does not exists!!')
    else:
        print(args['path'])
        FER_img(args['path'])

```

[FER_NET_RT.py] : 실시간 얼굴 표정 인식 / 103 lines

```
import cv2
import torch
import torchvision.transforms as transforms
from PIL import Image
import matplotlib.pyplot as plt
import argparse
import os
from model import *
import RPi.GPIO as GPIO
import time
from multiprocessing import Process

# GPIO - Board mode
output_dict = {'neutral':13, 'happiness' : 7, 'surprise': 29, 'sadness':40,
               'anger': 12, 'fear': 24, 'disgust':''}

# Load model
def load_trained_model(model_path):

    model = FER_Net()
    model.load_state_dict(torch.load(model_path, map_location=lambda
storage, loc: storage), strict=False)

    return model

# Recognition
def FER_Real_Time():

    # 1) Load model
    model = load_trained_model('./models/FER_trained_model.pt')

    emotion_dict = {0: 'neutral', 1: 'happiness', 2: 'surprise', 3:
'sadness',
                   4: 'anger', 5: 'disgust', 6: 'fear'}

    val_transform = transforms.Compose([
        transforms.ToTensor()])

    # 2) Set Camera
    cam_path = "nvarguscamerasrc ! video/x-raw(memory:NVMM), width=1920,
height=1080, format=(string)NV12, framerate=(fraction)30/1 ! nvvidconv flip-
method=2 ! video/x-raw, format=(string)BGRx ! videoconvert ! video/x-raw,
format=(string)BGR ! appsink"
    cam = cv2.VideoCapture(cam_path)
```

```

# Predict real-time Facial-Expression
while True:

    ret, frame = cam.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 3) Set Detector
    FER_cascade =
cv2.CascadeClassifier('./models/haarcascade_frontalface_default.xml')
    face = FER_cascade.detectMultiScale(frame)

    # Predict part
    for (x, y, w, h) in face:
        cv2.rectangle(frame, (x,y), (x+w, y+h), (255,0,0), 2)
        resize_frame = cv2.resize(gray[y:y + h, x:x + w], (48, 48))
        X = resize_frame/256
        X = Image.fromarray((X))
        X = val_transform(X).unsqueeze(0)

        # 4) Predict
        with torch.no_grad():
            model.eval()
            log_ps = model.cpu()(X)
            ps = torch.exp(log_ps)
            top_p, top_class = ps.topk(1, dim=1)
            pred = emotion_dict[int(top_class.numpy())]

        # 5) LED Control
        output_pin=output_dict[pred]
        print("Output Pin : ",output_pin)

        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(output_pin,GPIO.OUT,initial=GPIO.HIGH)
        print("Press CTRL+C when you want hte LED to stop blinking")

        time.sleep(0.5)
        GPIO.output(output_pin, GPIO.HIGH)
        print("LED ON")

        time.sleep(0.5)
        GPIO.output(output_pin, GPIO.LOW)
        print("LED OFF")

        GPIO.cleanup()

    cv2.putText(frame, pred, (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
(0, 255, 0), 1)

```

```
        print("Prediction : ",pred)

    # 6) Plot Prediction
    cv2.imshow('frame',frame)

    # Quit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# End
cam.release()
cv2.destroyAllWindows()

if __name__ == "__main__":

    FER_Real_Time()
```