포팅 매뉴얼

버전

BackEnd

- Spring boot : 2.7.14
- java : 17
- IntelliJ : Ultimate 23.1
- H2-Database : 2.2.220 (로컬, 테스트 환경)
- MariaDB : mariadb Ver 15.1 Distrib 10.3.38-MariaDB, for debian-linux-gnu (x86_64) using readline 5.2(배포 환경)
- Redis : 3.0.504(로컬, 테스트 환경), 7.0.12(배포 환경)

FrontEnd

- React : 18.2.0
- VScode: 1.18.1
- Node.js : 18.16.1
- Redux : 8.1.1
- npm : 9.5.1

Web RTC

- OpenVidu : 2.28.0

AI

- Teachable Machine
- Sequential Model
- 정규화 & Drop out
- Flask

협업 툴

- GitLab
- MatterMost
- Notion
- Jira

CI/CD

- AWS EC2
- Docker
- Nginx
- Jenkins

수동 배포

1. EC2에 docker 설치하기
   - `sudo apt update` `sudo apt upgrade` `sudo apt-get update` `sudo apt-get upgrade` 실행

```
# HTTPS를 통해 리포지토리를 사용할 수 있도록 패키지 인덱스를 업데이트 (apt)하고
# 패키지를 설치
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg

# Docker의 공식 GPG 키를 추가
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# 리포지토리를 설정
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

--------------------------------------------------------------------------------
# 패키지 index를 업데이트
sudo apt-get update

# Docker Engine, containerd 및 Docker Compose를 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# 이미지를 실행하여 Docker 엔진 설치가 성공했는지 확인
sudo docker run hello-world
docker -v
```

   - Docker network 설정
     - Docker Network란 같은 호스트 내에서 실행중인 컨테이너 간 연결을 할 수 있도록 돕는 논리적 네트워크 개념
     1. `docker network create [이름]` → 네트워크 생성
     2. `docker network ls` → 생성 확인
     3. docker 컨네이서 실행할 때 `--network [이름]` 옵션으로 연결하여 실행
     4. `docker network connect [네트워크 이름] [컨테이너 이름]` → 이미 동작중인 네트워크에 연결
2. OpenVidu 설치하기
   a. docker-compose 설치하기

```
# docker-compose 설치
sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/loca

# 다운로드한 docker-compose 파일을 실행 가능하도록 다운로드한 경로에 권한 부여
sudo chmod +x /usr/local/bin/docker-compose

# 심볼릭 링크 설정으로 path 경로를 아래와 같이 설정
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

# 설치 확인
docker-compose -v

# 설치 안될 시 설치
sudo apt-get install docker-compose
```

   b. OpenVidu 설치

```
sudo su # root 권한 주기
cd /opt
# 설치
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash
```

   c. OpenVidu 설정
   OpenVidu 객체 사용시 생성자에 들어가는 URL, SECRETKEY 설정 필요함
   URL ⇒ 서버 내 주소 ex https://i9c201.p.ssafy.io:4441 ⇒ 기본 포트번호는 443이지만 openvidu에서 제공하는 ngninx를 사용하지 않기 때문에 바꿈
   SECRET ⇒ 오픈 비두 기본 설치 /opt/openvidu/.env 파일 내 에서 설정

```
cd openvidu # 폴더 이동

# 환경 변수 설정 파일 수정
sudo vi .env

# 환경 설정 파일에서 DOMAIN_OR_PUBLIC_IP, OPENVIDU_SECRET 설정
DOMAIN_OR_PUBLIC_IP=i9c201.p.ssafy.io

OPENVIDU_SECRET=openvidu

CERTIFICATE_TYPE=letsencrypt

LETSENCRYPT_EMAIL= 이메일 주소

HTTP_PORT=4440

HTTPS_PORT=4441


# 오픈 비두 시작하기
./openvidu start
```

Database 설정

- EC2 서버에 직접 MariaDB, Redis를 설치해도 되지만 이번 프로젝트에선 모두 Docker Contatiner를 사용함
- **Mariadb**
  1. `docker pull mariadb` → mariadb image 다운로드 받기
  2. `docker run -d -p 3306:3306 --name mariadb --network {네트워크 이름} -e MARIADB_ROOT_PASSWORD=1234 mariadb` → docker 컨테이너 실행
     -d : 백그라운드에서 실행
     -p : 포트번호 설정, 앞→ 호스트의 포트번호 , 뒤→컨테이너 안에서의 포트번호
     --name : 컨테이너 이름
     -e : 기본 환경변수 설정
  3. `docker ps` → docker container 올라간 것 확인
  4. `docker exec -it [이름] mariadb -uroot -p` `컨테이너 띄울 때 설정한 비밀번호 입력` → Mariadb 컨테이너 접속
  5. `create database [DB_name];` ex) create database test; → 데이터베이스 설정
  6. `create user 'user_name'@'IP주소' identified by 'user_password';`
     ex) create user 'test'@'localhost' identified by 'test1234'; → 사용자 추가하기
  7. `grant all privileges on [DB_name].* to 'user_name'@'IP주소';`
     ex) grant all privileges on test.* to 'test'@'localhost'; → 사용자 권한 부여하기
  8. `flush privileges` → 변경 사항 적용
- **Redis**
  1. `docker pull redis` → redis image 다운로드 받기
  2. `docker run -d -p 6377:6377 --network {네트워크 이름}` `-v /redis/redis.conf:/etc/redis.conf --name redis redis` → redis 컨테이너 실행
  3. `docker ps` → docker container 올라간 것 확인
  4. `docker exec -it redis /bin/bash` → Redis 컨테이너 접속
  5. `redis-cli` → redis에 접속
  6. `info` → 버전 확인
  - 포트 번호 변경 → 마운트 한 redis.conf 아래와 같이 추가

```
# EC2 서버 내의 redsi.conf 파일로 이동

# 아래의 내용 추가

# 외부 접속 허용
bind 0.0.0.0
# 포트 번호 설정
port 6377
```

```
# 비밀번호 설정
requirepass refill1234
# 메모리 다 찾을 때 삭제할 알고리즘
maxmemory-policy volatile-ttl
```

FrontEnd & Backend 배포

1. git clone

```
git clone ~~~
```

2. FrontEnd 빌드 & 배포

```
# Dockerfile 있는 곳으로 이동
cd /frontend/refill

# .env 파일에 REACT...를 작성합니다.
echo "REACT_APP_KAKAO_API_KEY={키값}" > .env

# 프로젝트 빌드
npm install
npm run build

# Docker image 생성
docker build -t frontend .

# Docker 컨테이너 생성
docker run -d -p 3000:3000 --network docnet -e NODE_PATH=src -e TZ=Asia/Seoul --name frontend frontend
```

3. Backend 빌드 & 배포

```
# Dockerfile 있는 곳으로 이동
cd /backend/Refill

# 권한 설정
chmod +x gradlew

# 프로젝트 빌드
./gradlew clean bootjar

# Docker image 생성
docker build -t backend .

# Docker 컨테이너 생성
# 환경 변수 -e 옵션으로 추가
docker run -d -p 8090docker run -d -p 8090:8090 --network docnet -e AWS_S3_ACCESS_KEY={키값} -e AWS_S3_BUCKET_NAME={키값} -e AWS_S3_
```

Flask AI 모델 서버 배포

1. 플라스크 프로젝트 생성
2. requirements.txt 파일 추가

```
absl-py==1.4.0
astunparse==1.6.3
blinker==1.6.2
cachetools==5.3.1
certifi==2023.7.22
charset-normalizer==3.2.0
click==8.1.6
colorama==0.4.6
contourpy==1.1.0
cycler==0.11.0
filelock==3.12.2
Flask==2.3.2
flatbuffers==23.5.26
fonttools==4.41.1
fsspec==2023.6.0
gast==0.4.0
google-auth==2.22.0
google-auth-oauthlib==1.0.0
google-pasta==0.2.0
grpcio==1.56.2
h5py==3.9.0
huggingface-hub==0.16.4
```

```
idna==3.4
importlib-metadata==6.8.0
importlib-resources==6.0.0
itsdangerous==2.1.2
jax==0.4.14
Jinja2==3.1.2
joblib==1.3.1
keras==2.12.0
kiwisolver==1.4.4
libclang==16.0.6
Markdown==3.4.4
MarkupSafe==2.1.3
matplotlib==3.7.1
ml-dtypes==0.2.0
numpy==1.22.0
oauthlib==3.2.2
opencv-python==4.8.0.74
opt-einsum==3.3.0
packaging==23.1
Pillow==10.0.0
protobuf==4.23.4
pyasn1==0.5.0
pyasn1-modules==0.3.0
pyparsing==3.1.1
python-dateutil==2.8.2
PyYAML==6.0.1
regex==2023.6.3
requests==2.31.0
requests-oauthlib==1.3.1
rsa==4.9
safetensors==0.3.1
scikit-learn==1.2.2
scipy==1.11.1
six==1.16.0
tensorboard==2.12.3
tensorboard-data-server==0.7.1
tensorflow==2.12.0
tensorflow-estimator==2.12.0
tensorflow-intel
tensorflow-io-gcs-filesystem==0.31.0
termcolor==2.3.0
threadpoolctl==3.2.0
tokenizers==0.13.3
tqdm==4.65.0
transformers==4.31.0
typing_extensions==4.7.1
urllib3==1.26.16
Werkzeug==2.3.6
wrapt==1.14.1
zipp==3.16.2
```

3. app.py 작성

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.layers import Dropout
from PIL import Image, ImageOps
import cv2
import numpy as np
import pickle
import io

app = Flask(__name__)

class TrainingDropout(Dropout):
    def call(self, inputs, **kwargs):
        return super().call(inputs, training=True)

# Load the models
model = load_model('final_model.h5', custom_objects={'TrainingDropout': TrainingDropout}, compile=False)
teachable_model = load_model("keras_model.h5", compile=False)

# Load the label encoders
with open('label_encoder.pkl', 'rb') as f:
    label_encoder = pickle.load(f)

@app.route('/predict', methods=['POST'])
def predict():
    # Get the image from the request
    image_data = request.files['image'].read()

    # Convert the binary data to an image
    image = Image.open(io.BytesIO(image_data))
```

```
    # Convert the image to an array and preprocess for the original model
    image_array = img_to_array(image)
    image_array = cv2.resize(image_array, (224, 224))
    image_array = image_array / 255.0
    image_array = np.expand_dims(image_array, axis=0)

    # Make predictions using the original model
    predictions = []
    for i in range(100):
        predictions += [model.predict([image_array])]

    predictions = np.array(predictions)
    mean = np.mean(predictions, axis=0)
    var = np.var(predictions, axis=0)
    average_variance = np.mean(var)
    epsilon = 1e-7
    uncertainty = 0
    for i in range(7):
        uncertainty += np.log(mean[:, i] + epsilon) * mean[:, i]
    uncertainty = -uncertainty
    predicted_label = np.argmax(mean)
    class_label = label_encoder.inverse_transform([predicted_label])
    certainty = (1 - uncertainty) * 100
    certainty_from_variance = (1 - average_variance) * 100
    certainty_from_variance_rounded = round(certainty_from_variance, 1)

    size = (224, 224);
    teachable_image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)

    teachable_image_array = np.asarray(teachable_image)

    normalized_image_array = (teachable_image_array.astype(np.float32) / 127.5) - 1
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
    data[0] = normalized_image_array

    teachable_pre = teachable_model.predict(data)
    confidence_score = teachable_pre[0][0]
    confidence_score_percent = round(confidence_score * 100, 1)



    # # Process the image for the teachable machine model
    # teachable_image_array = np.asarray(image_array)  # 이미지 객체에서 배열로 바로 변환
    # normalized_teachable_image_array = (teachable_image_array.astype(np.float32) / 127.5) - 1
    # data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
    # data[0] = normalized_teachable_image_array
    #
    # # Make prediction using the teachable machine model
    # teachable_prediction = teachable_model.predict(data)
    # teachable_confidence_score = teachable_prediction[0][0]
    # teachable_confidence_score_percent = round(teachable_confidence_score * 100, 1)

    # Return the combined results
    result = {
        'result': class_label[0],
        'certainty': str(certainty_from_variance_rounded),
        'modelConfidence': str(confidence_score_percent)
    }

    return jsonify(result)

if __name__ == '__main__':
    app.run()
```

4. Dockerfile 작성

```
# 기본 이미지를 Python 3.9로 설정
FROM python:3.9.0rc2

# 작업 디렉토리 설정
WORKDIR /app

# 필요한 패키지 설치
RUN apt-get update && apt-get install -y \
    libgl1-mesa-glx \
    libsm6 \
    libxext6 \
    libxrender-dev

# 현재 디렉토리의 내용을 컨테이너의 작업 디렉토리에 추가
COPY . /app

# 파이썬 패키지 관리자 업그레이드
```

```
RUN python -m pip install --upgrade pip

# 필요한 파이썬 패키지 설치
RUN pip install --no-cache-dir -r requirements.txt

# 플라스크 환경 변수 설정
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV LD_LIBRARY_PATH=/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH

# 앱 실행
CMD ["flask", "run"]
```

5. 모델 추가
6. docker image 생성

```
# docker image 생성
docker build -f Dockerfile -t flask .

# Docker hub에 올리기 위해 image 이름 변경
docker tag flask {DockerHub 계정명}/flask:latest
```

7. Docker Hub에 올리기
   a. docker hub 계정 필요, docker Dosktop 실행

```
# Docker Hub에 이미지 올리기
docker push {Docker Hub 계정명}/flask:latest
```

8. EC2 서버에서 이미지 받기

```
# image 받기
docker pull {Docker Hub 계정명}/flask:latest
```

9. docker-compose.yml 파일 작성

```
version: "3"
services:
  flask1:
    image: {Docker Hub 계정명}/flask:latest
      ports:
        - "5000:5000"
  flask2:
    image: {Docker Hub 계정명}/flask:latest
      ports:
        - "5001:5000"
  flask3:
    image: {Docker Hub 계정명}/flask:latest
      ports:
        - "5002:5000"
```

10. docker 컨테이너 실행

```
docker-compose up -d
```

Nginx 설치 & SSL 인증

1. nginx 설치
   a. Nginx 설치 ⇒ `sudo apt install nginx`
   b. Nginx 상태 확인 ⇒ `sudo systemctl status nginx`
   c. Nginx 실행 시작 / 중지 ⇒ `sudo systemctl start nginx` `sudo systemctl stop nginx`
   d. Nginx 환경 설정 ⇒ `sudo vi /etc/nginx/sites-available/{파일명}.conf`
2. SSL 인증서 등록

a. let's Encrypt 설치 ⇒ `sudo apt-get install letsencrypt`

b. Certbot 설치 ⇒ `sudo apt-get install certbot python3-certbot-nginx`

c. Certbot 동작 ⇒ `sudo cerbot --nginx`

- 약관 동의 , 이메일, 서버 도메인 이름 필요

3. Nginx 설정 파일 작성

a. `sudo vi /etc/nginx/sites-available/{파일명}.conf`

```
# 로드 밸런싱을 위한 upstream 서버 블록
upstream flask_server {
        server localhost:5000;
        server localhost:5001;
        server localhost:5002;
}
#
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
}
server {
        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;
        server_name i9c201.p.ssafy.io; # managed by Certbot


        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                # try_files $uri $uri/ =404;
                proxy_pass http://localhost:3000;
        }

        location /api/ {
                proxy_pass http://localhost:8090;

                proxy_send_timeout 300s;
        }

        location /predict {
                proxy_pass http://flask_server;
        }
        listen [::]:443 ssl ipv6only=on; # managed by Certbot
        listen 443 ssl; # managed by Certbot
        ssl_certificate /etc/letsencrypt/live/i9c201.p.ssafy.io/fullchain.pem; # managed by Certbot
        ssl_certificate_key /etc/letsencrypt/live/i9c201.p.ssafy.io/privkey.pem; # managed by Certbot
        include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
server {
    if ($host = i9c201.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 ;
        listen [::]:80 ;
    server_name i9c201.p.ssafy.io;
    return 404; # managed by Certbot


}
server {
        listen 8088 ssl;
        server_name i9c201.p.ssafy.io;

        ssl_certificate /etc/letsencrypt/live/i9c201.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/i9c201.p.ssafy.io/privkey.pem;
```

```
        location / {
                proxy_pass http://localhost:8080;

        }
}
```

b. nginx 재시작

`sudo systemctl restart nginx`

SC-FEGAN 사용법

1. **아나콘다 설치**
   a. https://www.anaconda.com/products/individual 에서 최신 버전의 아나콘다 다운로드
   https://benn.tistory.com/26 를 참고
2. **가상 환경 생성 및 실행**
   a. 설치한 아나콘다에서 anaconda
3. **SC-FEGAN 사용환경 만들기**
   a. 해당 구글 드라이브에서 사전 모델을 다운받기 Google drive.
   b. https://github.com/run-youngjoo/SC-FEGAN Git Clone 하기
   c. 해당 프로젝트 폴더로 이동
   d. 구글 드라이브에서 받은 모델을 `ckpt라는 폴더` 를 만들고 폴더에 넣기
4. 초기 요구사항 설치
   a. `pip install -r requirements.txt`
5. 기타 설치
   a. `pip install PyQt5==5.12.3`
   b. `pip install numpy==1.16.1`
   c. `pip install opencv-python==4.0.0.21`
   d. `pip install pyyaml>=4.2b1`
   e. `pip install tensorflow==1.13.1`
   f. `pip install protobuf==3.6.1`
6. 실행
   a. `Python demo.py`

- 주의사항
  - 모델의 특성상 사진의 **파일명은 한글이 아니어야 하고 크기는 500 * 500이어야 함**
