기술문서

1. 개요

해당 프로젝트의 웹은 React와 FastAPI로 개발되었으며 각각의 git repository가 있습니다. 모델 파일은 FastAPI가 있는 git repository에 같이 있습니다.

git repository 주소:

- 프론트: https://github.com/leh60245/react-cloud
- 백: https://github.com/leh60245/fastapi

설치

프론트 프로그램을 사용하려면 우선 의존성 모듈을 설치해야 합니다.

yarn

 # or

yarn install

실행

프론트 프로그램을 실행하는 방법입니다.

npm start

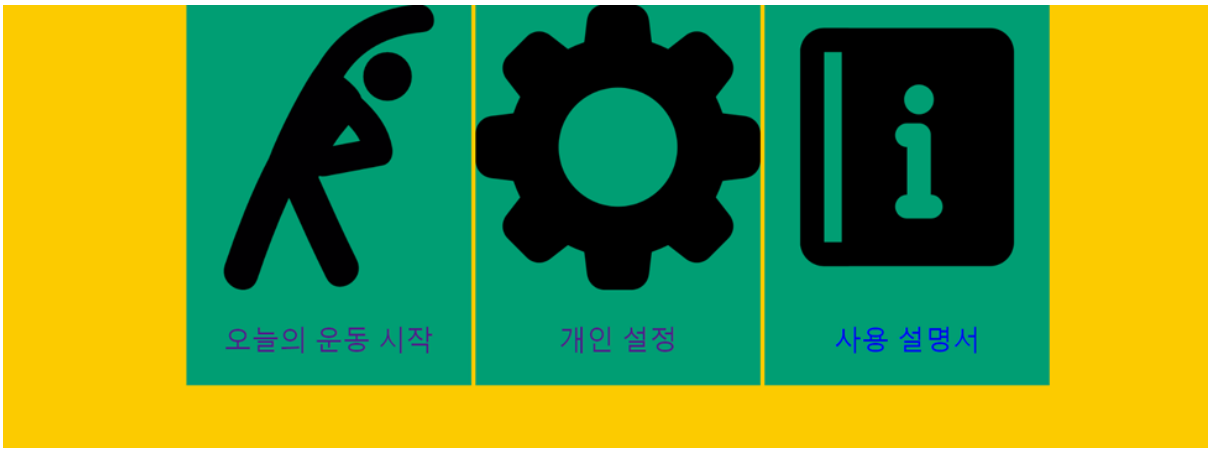
백 프로그램을 실행하는 방법입니다.

uvicorn main:app --reload

화면 구성

화면은 크게 4가지로 “첫 화면”, “오늘의 운동 화면”, “개인 설정 화면”, “사용 설명서 화면” 입니다.

- 첫 화면: 프로그램이 시작되면 가장 먼저 보이는 화면으로 “오늘의 운동 시작”, “개인 설정”, “사용 설명서” 3개의 버튼이 있습니다. 각 버튼을 누르면 해당 화면으로 전환이 됩니다.



- 오늘의 운동 화면: 첫 화면에서 보이는 “오늘의 운동 시작” 버튼을 누르면 나오는 화면으로 운동 자세 피드백을 음성으로 받을 수 있습니다. 화면의 구성은 좌측, 중앙, 우측으로 나누어지며 좌측에는 카메라가 신체를 인식하는 정도를 0~100% 사이의 수치로 알려줍니다. 중앙에는 카메라로 찍히는 사용자의 모습을 보여줍니다. 우측에는 진행된 단계를 보여줍니다.

- 개인 설정 화면: 첫 화면에서 보이는 “개인 설정” 버튼을 누르면 나오는 화면으로 프로그램의 배경색을 변경할 수 있습니다.

- 사용 설명서 화면: 첫 화면에서 보이는 “사용 설명서” 버튼을 누르면 나오는 화면으로 처음 프로그램을 사용하는 사용자를 위한 설명서로 제작 예정입니다. 설명할 내용은 “오늘의 운동 시작” 페이지의 화면 구성입니다.

2. 코드 리뷰

백엔드와 프론트엔드, 마지막으로 모델 순으로 코드 리뷰를 하겠습니다.

2-1. 백엔드

사용된 파일은 크게 3가지입니다.

- main.py: 프론트 서버에서 날라온 이미지를 모델로 돌리고 예측한 결과를 반환합니다.

- i_model.h5: 직접 생성한 모델입니다. (2023.12.13.수 기준)

- i_model.ipynb: 사용할 모델을 만드는 파일입니다.

모델을 제외한 main.py를 다루겠습니다.

main.py

```
from fastapi import FastAPI, HTTPException
from io import BytesIO
from PIL import Image
import base64
import numpy as np
from pydantic import BaseModel
from tensorflow.keras.models import load_model
from starlette.middleware.cors import CORSMiddleware
```

필요한 라이브러리를 설치합니다.

```
app = FastAPI()

origins = [
    "http://localhost:3000",    # 또는 "http://localhost:5173"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

백엔드 서버를 만드는데 필요한 설정을 작성합니다.

```python
class ImageData(BaseModel):
    image_str: str

model = load_model("i_model.h5", compile=False)

@app.post("/uploadfile/")
async def create_upload_file(image_data: ImageData):
    try:
        # 원본 이미지를 image에 저장합니다.
        header, encoded = image_data.image_str.split(",", 1)
        image_bytes = base64.b64decode(encoded)
        image = Image.open(BytesIO(image_bytes)).convert('RGB')

        # 모델에 맞는 인풋 형태로 변형합니다.
        image = image.resize((64, 64))
        image_array = np.array(image) / 255.0
        image_array = np.expand_dims(image_array, axis=0)

        # 이미지를 예측하고 결과를 반환합니다.
        # 반환하는 결과는 1. 예측한 클래스: int, 2. 각 클래스별 예측한 값: List<int>
        result = model.predict(image_array)
        result_list = result.tolist()[0]
        class_0_prob = result_list[0]
        class_1_prob = result_list[1]
        class_2_prob = result_list[2]
        if 0.31 <= class_0_prob < 0.35 and class_1_prob < 0.385 and class_2_prob < 0.316:
            predicted_class = 0
        elif 0.3 <= class_0_prob < 0.355 and 0.35 <= class_1_prob and 0.26 <= class_2_prob < 0.325:
            predicted_class = 1
        elif 0.29 <= class_0_prob < 0.355 and 0.36 <= class_1_prob < 0.39 and 0.28 <= class_2_prob < 0.32:
            predicted_class = 2
        else:
            predicted_class = -1
        return {"predicted_class": predicted_class, "acc_arr": result_list}
    except Exception as e:
        raise HTTPException(status_code=400, detail=f"이미지 처리 실패: {str(e)}")
```

프론트 서버에서 날라온 이미지를 직접 만든 모델로 예측을 한 뒤 예측한 클래스와 각 클래스별 예측한 값을 반환합니다.

2-2. 프론트엔드

웹 화면을 구성하는 파일과 데이터는 모두 src 폴더 내부에 있습니다.

App.js: “첫 화면”, 즉 웹 앱의 메뉴를 구성하는 js 파일입니다. 메뉴의 아이콘,종류,구성 요소를 변경,추가,삭제하고자 하거나 메뉴의 음성 안내 과정에서 바꿀 것이 있다면 해당 파일을 수정하면 됩니다.

routes 폴더: 버튼을 눌렀을 때 나오는 화면과 json 형식의 데이터가 있습니다.

- exercise.jsx: “오늘의 운동 화면”을 구성합니다.
- setting.jsx: “개인 설정 화면”을 구성합니다.
- instruction.js: “사용 설명서 화면”을 구성합니다.
- during_exercise.json: 운동을 시작하고 스텝별로 안내드릴 내용과 그 밖의 상황이 일어났을 때 안내드릴 내용이 텍스트 형식으로 저장돼 있습니다.
- keypoints_korea_name.json: MoveNet이 검출하는 keypoints와 대응되는 한국 이름이 저장돼 있습니다.

utils 폴더: 화면을 구성하는데 필요한 여러 함수들을 모아둡니다.

- getSpeech.jsx: 음성 안내에 필요한 함수가 있습니다.

1.Src/App.js

아래와 같이 오늘의 운동 시작(exercise), 개인 설정(setting), 사용설명서(instruction) 메뉴에 해당하는 아이콘 이미지와 json 파일 경로를 불러오고, 음성 안내를 구성하는 getSpeech파일을 import 해옵니다.

```
// Routes
import Exercise from "./routes/exercise";
import Setting from "./routes/setting";
import Instruction from "./routes/instruction";

// Utils
import { getSpeech } from "./utils/getSpeech";

// img
import exerciseImg from "./src/stretching-exercises.png";
import settingImg from "./src/settings.png";
import instruction from "./src/guidebook.png";

import "./App.css";
```

1-1) function Home()

애플리케이션의 메인 페이지를 구성하는 함수입니다.

```
const [hover, setHover] = useState("");
```

먼저 현재 호버(마우스와 메뉴 요소의 상호 작용) 상태를 useState를 사용하여 관리합니다.

```
const menuList = [
  {
    id: "exercise",
    link: "/exercise",
    img: exerciseImg,
    icorn: "_Icon_exercise",
    text: "오늘의 운동 시작",
  },
  {
    id: "setting",
    link: "/setting",
    img: settingImg,
    icorn: "_Icon_setting",
    text: "개인 설정",
  },
  {
    id: "instruction",
    link: "/instruction",
    img: instruction,
    icorn: "_Icon_instruction",
    text: "사용 설명서",
  },
];
```

다음으로 오늘의 운동 시작(exercise), 개인 설정(setting), 사용설명서(instruction) 3개의 메뉴에 대해 각 id 별로 링크, 아이콘, 아이콘 이미지 제목(icorn), 표시할 텍스트와 같은 속성을 지정하여 메뉴 객체를 담은 'menuList' 배열을 정의합니다.

```
// 마우스가 메뉴 위에 있을 때 음성 재생
const handleMouseEnter = (text) => {
  getSpeech(text);
  setHover(text);
};

// 마우스가 메뉴에서 벗어날 때 음성 중단
const handleMouseLeave = () => {
  if (window.speechSynthesis && window.speechSynthesis.speaking) {
    window.speechSynthesis.cancel();
  }
  setHover("");
};

// 포커스가 메뉴 위에 있을 때 음성 재생
const handleFocus = (text) => {
  getSpeech(text);
  setHover(text);
};

// 포커스가 메뉴에서 벗어날 때 음성 중단
const handleBlur = () => {
  if (window.speechSynthesis && window.speechSynthesis.speaking) {
    window.speechSynthesis.cancel();
  }
  setHover("");
};
```

마우스의 클릭, 커서 움직임에 반응하는 이벤트 핸들러 onFocus(클릭 시), onMouseEnter, onMouseLeave에 의해 이벤트가 발생했을 때 위에서 정의한 호버 상태가 변경되고

텍스트를 음성으로 변환하는 getSpeech를 불러와 메뉴 이름을 음성으로 안내할 수 있도록 합니다.

```
  const linkMenuList = menuList.map((menu) => (
    <Link
      key={menu.id}
      to={menu.link}
      style={{ textDecoration: "none" }}
      onMouseEnter={() => handleMouseEnter(menu.text)}
      onMouseLeave={handleMouseLeave}
      onFocus={() => handleFocus(menu.text)}
      onBlur={handleBlur}
    >
      <Box w="100%" h="100%" border="20px" backgroundColor="#009E73">
        <Card maxW={{ base: "100%", sm: "200px" }}>
          <CardHeader>
            <Image w="100%" h="100%" id="image" src={menu.img} />
          </CardHeader>
          <CardBody>
            <Text fontSize="50px">{menu.text}</Text>
          </CardBody>
        </Card>
      </Box>
    </Link>
  ));

  return (
    <Center>
      <HStack>{linkMenuList}</HStack>
    </Center>
  );
}
```

마지막으로 저장된 메뉴의 배열로부터 각 메뉴의 속성을 불러와 UI를 렌더링합니다. 먼저 Box를 생성하여 w,h(너비,높이), 외곽선 두께(border), 색깔(backgroundColor) 의 속성을 지정해 줍니다. 다음으로 메뉴 아이콘 이미지를 헤더에, 아이콘 이름을 body로 하는 카드를 넣어줍니다. 수평 방향으로 메뉴를 배치하기 위해 HStack 컴포넌트를 사용합니다.

1-2) function App()

App.js 파일의 메인 컴포넌트로, 첫 화면이 구성될 때 구조와 라우팅을 정의합니다.

메뉴 요소 (현재 운동 시작(exercise), 개인 설정(setting), 사용설명서(instruction) 3개)와 각각의 메뉴를 구현하는 jsx 파일이 저장된 경로를 이어줍니다.

2. src/routes/exercise.jsx

오늘의 운동 화면을 구성하는 코드입니다. 코드에서는 크게 커스텀 학습 모델을 활용하기 위해 이미지를 서버에 전송하는 함수, MoveNet을 활용하여 사용자의 현재 포즈를 감지하는 함수, 포즈의 정확도를 계산하는 함수, 그리고 단계별로 운동을 진행시키는 함수로

구성되어 있습니다. during_exercise.json 으로부터 각 단계와 단계별로 안내할 음성 안내의 내용을 , keypoints_korea_name.json 으로부터 각 keypoint와 keypoint에 대응하는 한글 이름을 import합니다.

2-1) const sendImageToServer

```
// 이미지를 서버에 전송하는 함수
const sendImageToServer = async (imageSrc) => {
  try {
    const response = await fetch("http://localhost:8000/uploadfile/", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ image_str: imageSrc }),
    });
    // 서버로부터 응답 처리
    const data = await response.json();
    return data;
  } catch (error) {
    console.error("Error:", error);
    return null;
  }
};
```

이 함수는 저희 팀에서 자체적으로 만든 커스텀 모델을 활용하기 위해 localhost:8000(FastAPI 서버)로 이미지를 보내는 과정에서 async 키워드를 사용하여 비동기 작업을 수행하며, fetch 함수를 통해 RESTful API 중 POST 메소드를 사용하여 로컬 FastAPI 서버의 경로로 요청을 보냅니다.

성공적으로 서버에서 이미지 데이터를 모델로 처리해서 응답을 반환하면 그 값을 data로 리턴해줍니다.

2-2)const detectPose

```
const detectPose = async (net, webcamRef) => {
  if (
    typeof webcamRef.current !== "undefined" &&
    webcamRef.current !== null &&
    webcamRef.current.video.readyState === 4
  ) {
    const video = webcamRef.current.video;
    const poses = await net.estimatePoses(video);
    return poses;
  }
  return null;
};
```

위에서 커스텀 모델을 활용했던과 달리 이 함수는 기존 모델인 MoveNet을 활용해 keypoint의 상태를 검출해내기 위한 함수입니다. 먼저 webcamRef(웹캠 컴포넌트 참조)에서 웹캠의 현재 값이 정의되어 있고, readyState가 4, 즉 웹캠에서 보내는 미디어 데이터가 충분한 상태인지 확인하는 조건문을 확인합니다. 조건문이 참이면 net(Pose Detection 모델).estimatePoses 메소드를 이용해 감지된 포즈 데이터를 poses에 저장해 리턴해줍니다.

2-3) function Exercise

이 함수는 앞서 설명한 함수를 활용해 운동 과정을 단계별로 구현하고, 오늘의 운동 화면을 구성하는 exercise.jsx 파일의 중심 요소입니다.

단계 구성의 로직은 다음과 같습니다.

1.현재 단계에 해당하는 운동에 대해 음성 안내를 해줍니다. 만약 마지막 스텝이라면 운동을 종료합니다.

2.5초 대기합니다.

3.위 2-2)에서 설명한 MoveNet을 이용해 필요한 keypoints가 모두 감지되었는지 확인합니다. 모두 감지되지 않았다면, 보이지 않는 부위가 있다고 음성 안내하며 필요한 키포인트가 모두 감지될 때까지 반복합니다.

4.만약 모델을 체크하는 단계라면, FastApi 서버로 이미지를 보내 응답을 받아옵니다. 응답 결과 사용자가 제대로 자세를 취하고 있지 않다면, 음성 안내하고 제대로 자세를 취할때까지 반복합니다.

5. 다음 스텝으로 넘어갑니다.

단계의 처리는 118행의 const processCurrentStep 함수가 처리합니다.

```
const processCurrentStep = async () => {
  if (isProcessing) return;
  setIsProcessing(true);
  const step = stepsData[currentStep];
```

만약 대기 시간을 변경하고자 한다면 126행의 setTimeout 메소드를 수정하면 됩니다.

```
// 5초 대기
await new Promise((resolve) => setTimeout(resolve, 5000));
```

keypoint 감지는 아래 조건문을 이용해 처리합니다.

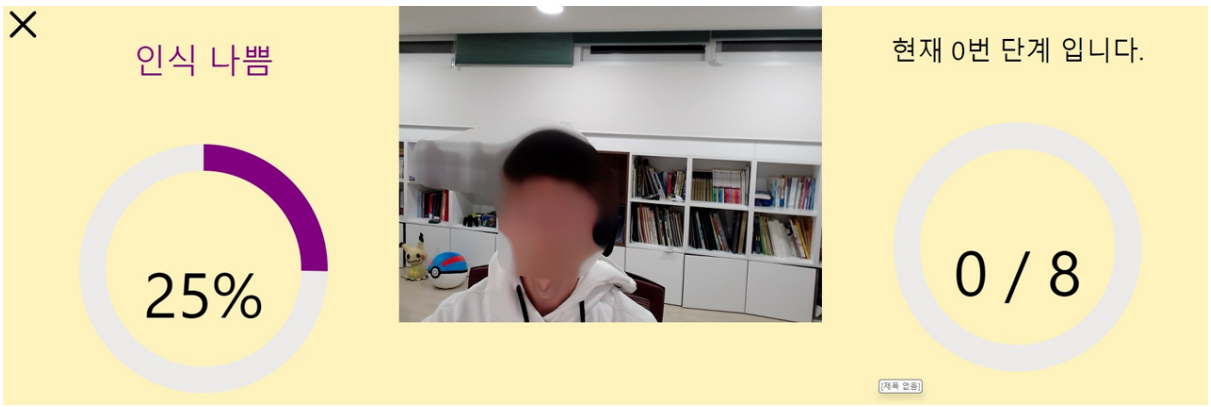
```
if (now_pose === null) {
  // 전혀 사람이 보이지 않거나 카메라가 사람 일부분도 찾지 못할 때.
  await speakText(duringExerciseData.cannot_recognize[2].recognize);
  await new Promise((resolve) => setTimeout(resolve, 5000));
  setIsProcessing(false);
  return;
} else if (undetectedKeypoints.length > 0) {
  // 감지되지 않은 keypoints에 대한 음성 메시지
  const message = `보이지 않는 부위가 있습니다: ${undetectedKeypoints.join(
    ", "
  )}입니다. 다시 조정 바랍니다.`;
  await speakText(message);
  await new Promise((resolve) => setTimeout(resolve, 5000));
  setIsProcessing(false);
  return;
}
```

모델을 사용할 경우 step.use_model 값이 true인지 확인한 다음 앞서 정의한 sendImageToServer함수를 이용해 fastAPI 서버로 웹캠 이미지를 보내줍니다.

```
      // 모델 사용 여부
      if (step.use_model) {
        // 이미지를 서버에 보내고 결과를 받음
        await new Promise((resolve) => setTimeout(resolve, 3000));
        const imageSrc = captureImage(webcamRef);
        const response = await sendImageToServer(imageSrc);
        if (
          step.class !== response.predicted_class &&
          response.acc_arr[step.class] < thresholdByClass[step.class]
        ) {
          // 예측한 class가 다르고, 실제로 일정 임계값 보다 정확도가 작다면 틀린 자세로 판단
          await speakText(duringExerciseData.check[0].recognize);
          await new Promise((resolve) => setTimeout(resolve, 5000));
          setIsProcessing(false);
          return;
        } else {
          // 자세가 옳바를 때
          await speakText(duringExerciseData.check[1].recognize);
          await new Promise((resolve) => setTimeout(resolve, 5000));
        }
      }

      setCurrentStep((current) => current + 1);
    }
    setIsProcessing(false);
  };
```

마지막으로 화면 구성은 233행부터 return 값을 정의하여 해줍니다.



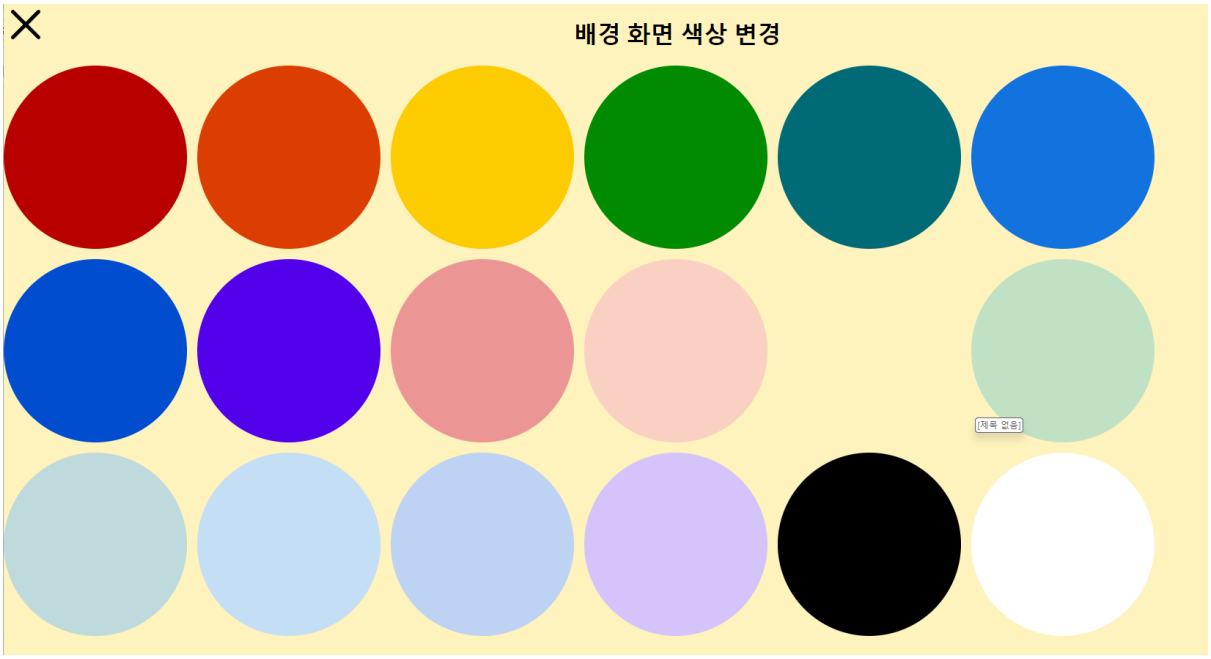
화면 구성은 SimpleGrid를 사용하여 포즈의 정확도, 웹캠, 현재 단계를 나타내는 3개의 열로 구성되어 있습니다.

각 열은 Box 컴포넌트에 요소를 집어넣는 방법으로 구성되었으며, 첫번째 열은 포즈의 정확도를 CircularProgress가 poseAccuracy를 가져와 나타냅니다. 먼저 CloseIcon을 설정해 handleGoBack 함수를 이용하여 좌측 상단에 X표시를 누르면 홈 화면으로 돌아가게 합니다.

인식률이 나쁠때 색깔이 바뀌게 걸정되어 있으며 '나쁘다'의 기준은 critical_point 값이 결정합니다. 이 변수는 87행에서 정의되며 기본값은 40입니다.

웹캠의 경우 264행의 transform:"scaleX(-1)"로 좌우 반전이 적용되어 있으며 화면의 높이와 너비 값을 변수를 변경해 조정할 수 있습니다.

3. src/routes/setting.jsx



이 코드는 "개인 설정" 메뉴를 구성하는 코드로, 직관적이고 큼지막한 UI를 통해 시각장애인 사용자의 특성을 고려하여 테마 색상을 설정할 수 있도록 합니다.

```
import React, { useState } from "react";
import { CirclePicker } from "react-color";
import { useNavigate } from "react-router-dom";

import { Heading, Container } from "@chakra-ui/react";
import { CloseIcon } from "@chakra-ui/icons";

```

ui는 react-color의 CirclePicker를 불러와서 사용합니다. 해당 색상 선택기는 react-color 라이브러리에서 유일하게 각 색상 선택별 크기 조정이 가능하여 사용되었습니다.

```
<CirclePicker
  width="100%"
  circleSize={250}
  colors={[
    "#B80000",
    "#DB3E00",
    "#FCCB00",
    "#008B02",
    "#006B76",
    "#1273DE",
    "#004DCF",
    "#5300EB",
    "#EB9694",
    "#FAD0C3",
    "#FEF3BD",
    "#C1E1C5",
    "#BEDADC",
    "#C4DEF6",
    "#BED3F3",
    "#D4C4FB",
    "#000000",
    "#FFFFFF",
  ]}
```

색상 아이콘의 크기와 색상의 종류는 circlesize 변수와 colors 리스트를 수정하여 바꿀 수 있습니다.

2-3. 모델

1. 소개 : 기술문서의 모델 부분에서는 딥러닝 모델의 성능평가에 대한 결과를 제시합니다. 모델은 주어진 이미지 데이터셋을 기반으로 학습되었으며, 테스트 데이터에 대한 정확도와 예측 결과를 분석합니다.

2. 코드 분석

- 데이터 로딩 및 전처리

```
# 이미지 데이터 로딩
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(64, 64),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(64, 64),
    batch_size=batch_size,
    class_mode='categorical')
```

-

: ImageDataGenerator를 사용하여 이미지 데이터를 전처리하고, flow_from_directory 함수를 통해 디렉토리에서 바로 데이터를 불러오고 전처리합니다.
: 학습 및 검증 데이터셋은 64x64 크기로 리사이즈되며, categorical 모드로 클래스 레이블을 생성합니다.

- 모델 아키텍처

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

-

: Sequential 모델을 사용하여 CNN 모델을 구성합니다.
: Convolutional 레이어와 MaxPooling 레이어를 번갈아가며 쌓아가는 구조입니다.
: Flatten 레이어를 통해 2D 텐서를 1D로 평탄화하고, Fully Connected 레이어를 추가합니다.

: 마지막 레이어는 3개의 클래스를 분류하기 위한 softmax 활성화 함수를 사용합니다.

- 모델 컴파일 및 학습

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_s
```

- : 손실 함수로 categorical_crossentropy를, 옵티마이저로 adam을 사용하여 모델을 컴파일합니다.
  : fit 함수를 통해 데이터를 사용하여 모델을 학습합니다.

- 모델 평가

```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(64, 64),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False)

test_loss, test_accuracy = model.evaluate(test_generator, st
```

- : 테스트 데이터에 대해 'evaluate' 함수를 통해 손실과 정확도를 계산합니다.

- 예측 및 결과 분석

```
predictions = model.predict(test_generator, steps=test_generator.sample
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

confusion_mtx = confusion_matrix(true_classes, predicted_classes)
classification_report_result = classification_report(true_classes, pred

# 클래스별 예측 결과 및 틀린 예측 샘플 확인
for i, image_path in enumerate(test_generator.filepaths):
    img = load_img(image_path, target_size=(64, 64))
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0

    predicted_class = class_labels[predicted_classes[i]]
    true_class = class_labels[true_classes[i]]

    # 예측이 틀린 경우에 대해서만 출력
    if predicted_class != true_class:
        print(f"Image: {os.path.basename(image_path)}, Predicted Class:
```

: predict 함수를 통해 테스트 데이터에 대한 예측을 수행 : 혼동 행렬, 분류 보고서를 생성하고 클래스별로 예측이 틀린 샘플을 확인합니다.

- 모델 저장
- ```
  model.save('/content/drive/MyDrive/Vision_Pro/i_model.h5')
  ```

: 학습이 완료된 모델을 저장합니다.

3. 모델 개요

- 모델 이름 : 'i_model.h5'
- 아키텍쳐 : convolutional neural network(CNN)
- 층의 구성
  - Conv2D 층 3개
  - MaxPooling2D 층 3개
  - Flatten 층 - Dense 층 2개
  - Dropout 층 1개

4. 학습 과정

- 총 Epochs : 7
- 최종 학습 정확도 : 78.09%
- 최종 검증 정확도 : 76.62%
  ```

5. 테스트 결과

- 테스트 loss : 0.9827
- 테스트 Accuracy : 76.62%

6. 초기 모델 성능 분석

- Confusion Matrix

```
Confusion Matrix:
[[  0  42 255]
 [220  21  45]
 [  5 299   4]]
```

: class_1에 대한 대다수의 예측이 class_3로 오인되었습니다.
: class_2의 경우 class_3로 오인된 경우가 많았습니다.
: class_3의 경우 class_2로 오인된 경우가 많았습니다.

- Classification report

```
Classification Report:
              precision    recall  f1-score   support

    step_3_3       0.00      0.00      0.00       297
    step_1_3       0.06      0.07      0.06       286
    step_2_3       0.01      0.01      0.01       308

    accuracy                           0.03       891
   macro avg       0.02      0.03      0.03       891
weighted avg       0.02      0.03      0.03       891
```

: 여기서는 각 클래스에 대한 정밀도, 재현율, F1-score 등의 지표를 확인 가능합니다.
: class_3의 경우 정확도가 낮아, 해당 클래스를 식별하는 데 어려움이 있음을 시사합니다. class_1, class_2 역시 성능이 향상되어야 했습니다.

7. 예측 결과

- 랜덤 예측 결과

```
1/1 [==============================] - 0s 35ms/step
    Image: ubusaugmented4.png, Predicted Class: class_1
1/1 [==============================] - 0s 32ms/step
    Image: ubusaugmented7.png, Predicted Class: class_1
```

- 특정 이미지 예측 결과

```
1/1 [==============================] - 0s 94ms/step
Predicted Class: class_1
Predictions: [0.296292   0.38670206 0.31700587]
Second Highest Probability : 0.31700587272644043
```

- 평가 및 추가조치

: 초기 모델은 전체적인 평가 matrix를 통해 모델이 특정 클래스에 대해 예측을 잘 수행하지 못하고 있는 것으로 나타났습니다. 클래스별로 예측 결과가 부족하거나 불균형하게 나타나 있어, 추가적인 데이터 수집이나 클래스 간 데이터 균형을 맞추는 데이터 전처리가 필요했습니다. 따라서, 원래의 데이터셋에 팀원들의 데이터를 추가하여 다시 한 번 학습을 진행했고, 웹캠에서 들여온 이미지 처리에서 수행되는 전처리로 모델의 성능을 높이고자 했습니다.

8. 모델 저장 : 학습된 모델은 'i_model.h5'로 저장됩니다.

9. 이미지 평가(밑에는 출력 결과의 예시)
- 랜덤 이미지
  : True Label : step_1_3
  : predicted Label : step_3_3

- 특정 이미지
  : Predicted Class = step_3_3
  : Predictions = [0.296292, 0.38670206, 0.31700587]
  : Second Highest Probability = 0.3170

10. 코드 실행 환경 : 코드는 Google Colab 상에서 실행하여 모델 학습 및 테스트를 진행하였습니다.

11. 결론 : 모델은 주어진 데이터에 대해 일정 수준의 성능을 보였고, 추가 수정 및 학습을 통해 성능을 향상시켰습니다. 일부 클래스에 대한 예측 정확도가 낮은 것으로 초기에는 분석되었으나 추가적인 데이터 수집 및 모델 튜닝을 통해 학습과 테스트 성능이 향상되었다. 처음의 70%정도의 테스트 정확도를 보였고 이후 학습이 추가로 진행됐을 때는 90~95% 사이의 정확도를 보였습니다.

2-4. 카메라 분할**(**웹캠 사용**)**

1. 코드 분석
- 웹캠 액세스를 위한 JavaScript 코드

```
def start_webcam():
    js = Javascript('''
        async function startWebcam() {
            const video = document.createElement('video');
            const stream = await navigator.mediaDevices.getUserMedia({
            video.srcObject = stream;

            // 비디오 로딩 대기
            await video.play();

            // 비디오 스트림에서 프레임 캡처
            const canvas = document.createElement('canvas');
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext('2d').drawImage(video, 0, 0);

            // 캡처한 프레임을 JPEG 데이터 URL로 반환
            return canvas.toDataURL('image/jpeg', 0.8);
        }
    ''')
    display(js)
```

: 비디오 요소를 생성하고, 웹캠 스트림에서 프레임을 캡쳐한 후 JPEG 데이터 URL로 반환합니다.

- 이미지 캡처 및 처리

```
def capture_webcam():
    data_url = eval_js('startWebcam()')
    binary_data = b64decode(data_url.split(',')[1])
    image = Image.open(io.BytesIO(binary_data))

    # 이미지 모드가 RGBA(투명도)인 경우 RGB로 변환
    if image.mode == 'RGBA':
        image = image.convert('RGB')

    return image
```

: 이 함수를 JavaScript 함수를 호출해 웹캠 이미지를 호출하고, base64로 인코딩된

데이터를 디코딩하여 PIL 이미지 객체로 변환합니다. 이미지가 RGBA 모드인 경우 RGB로 변환합니다.

- 웹캠 초기화 및 이미지 캡처/저장

```
# 웹캠 시작
start_webcam()

# 이미지 캡처
captured_image = capture_webcam()
```

```
# 이미지를 JPEG 파일로 저장
captured_image.save('captured_image.jpg', 'JPEG')

# 저장된 이미지 표시
display(IPImage(filename='captured_image.jpg'))
```

: start_webcam() 함수를 사용하여 웹캠을 초기화하고 capture_webcam() 함수를 사용하여 웹캠에서 단일 프레임을 캡처 : 캡처한 이미지는 'captured_image.jpg'라는 이름으로 JPEG 파일로 저장되고, 그 후 IPython의 display 함수를 사용하여 저장된 이미지가 표시됩니다

- preprocess_webcam_image 함수

```
def preprocess_webcam_image(img):
    img = img.resize((image_height, image_width))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # 정규화

    return img_array
```

: 이미지를 모델에 입력하기 전에 필요한 전처리 단계를 수행하여 모델이 이해할 수 있는 형태로 변환합니다.

2. 코드 목적 : Google Colaboratory(Colab) 환경에서 웹캠을 사용하여 이미지를 캡처하고, TensorFlow와 Keras를 이용하여 미리 훈련된 신경망 모델을 활용하여 이미지를 분류하는 작업을 수행합니다.

3. 코드 실행 흐름

1) Colab에서 웹캠을 사용하기 위한 JavaScript 코드를 실행하고 웹캠을 초기화

2) 웹캠에서 이미지를 캡처하고, 해당 이미지를 전처리
3) 전처리된 이미지를 훈련된 모델에 입력하여 클래스 예측을 수행
4) 예측된 클래스를 출력하여 사용자에게 표시합니다.

4. 역할 및 기능

- JavaScript 코드: Colab 환경에서 웹캠을 사용하여 이미지를 캡처하고 이를 Python으로 가져옵니다.
- capture_webcam 함수: JavaScript를 통해 캡처된 이미지를 가져오고, PIL Image로 변환합니다.
- preprocess_webcam_image 함수: 이미지를 정규화하고 모델에 입력하기 적합한 형식으로 전처리해줍니다.
- 모델 예측: 전처리된 이미지를 사용하여 사전 훈련된 모델로 클래스 예측을 수행합니다.
- 결과 출력: 예측된 클래스를 출력하여 사용자에게 표시합니다.