

Skin-Check

프로젝트

*

AI & 빅데이터 분석
활용 풀스택 서비스 개발

2조

박은택 김유하 서현세 손수현

목차

* 01

프로젝트 배경

프로젝트 배경

프로젝트 구성도

* 02

모델 / 데이터 분석

데이터 수집

모델 정의서

머신러닝 모델 학습

분석 / 예측

* 03

서비스 구현

개발환경 구축 및 서비스 개발

* 04

개선사항

프로젝트 배경

프로젝트 배경

개인 맞춤형 피부 관리에 대한 수요 증가

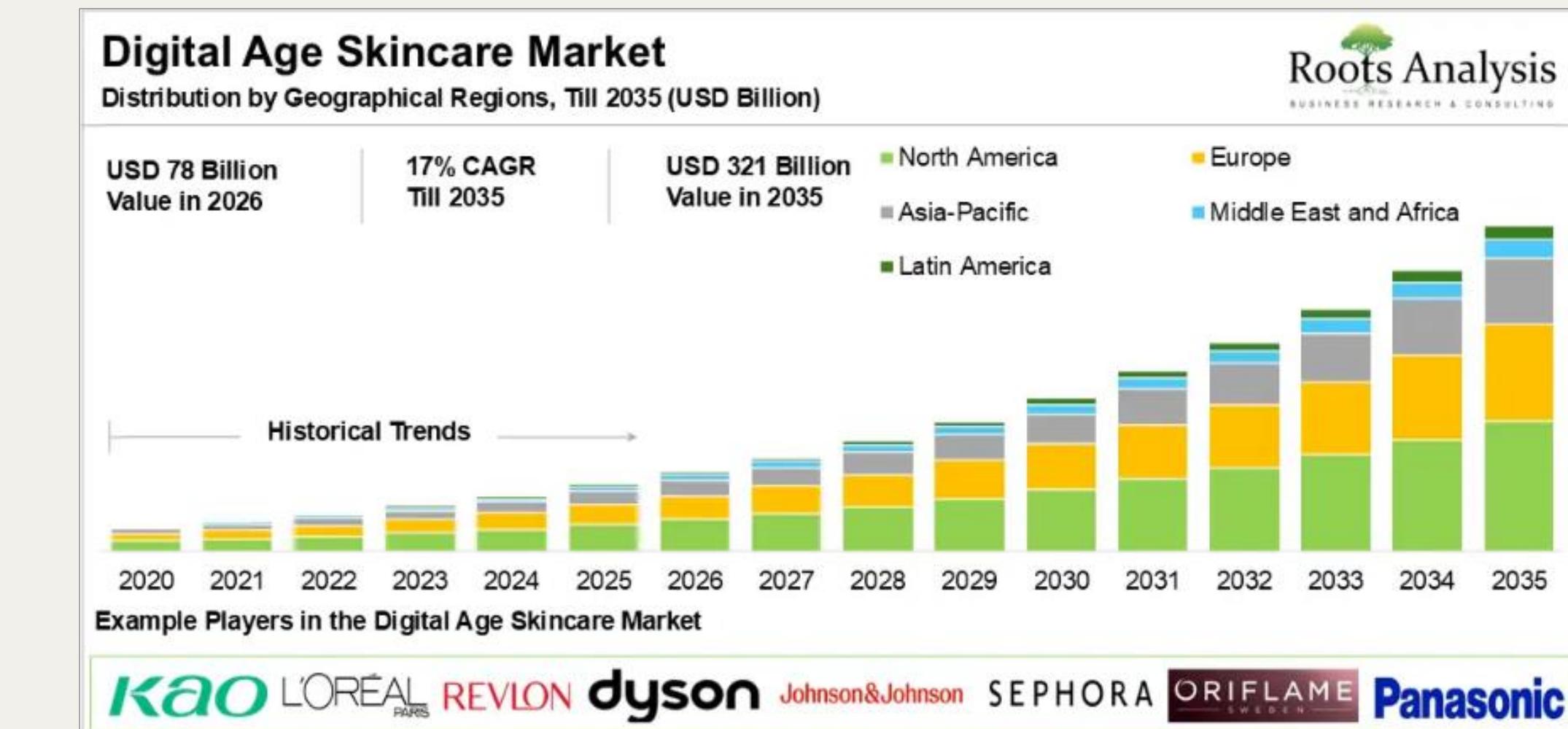
- 최근 비대면 환경 확산과 함께 모바일 기반 피부 분석 서비스에 대한 수요가 증가하고 있다.

접근성 문제

- 기존 피부 진단은 전문가 상담이나 고가의 장비에 의존하는 경우가 많아 접근성과 객관성 측면에서 한계가 존재한다.

객관성 문제

- 개인의 피부 상태는 주관적인 판단에 의존하는 경우가 많아, 정량적이고 자동화된 분석 방식의 필요성이 대두되고 있다.

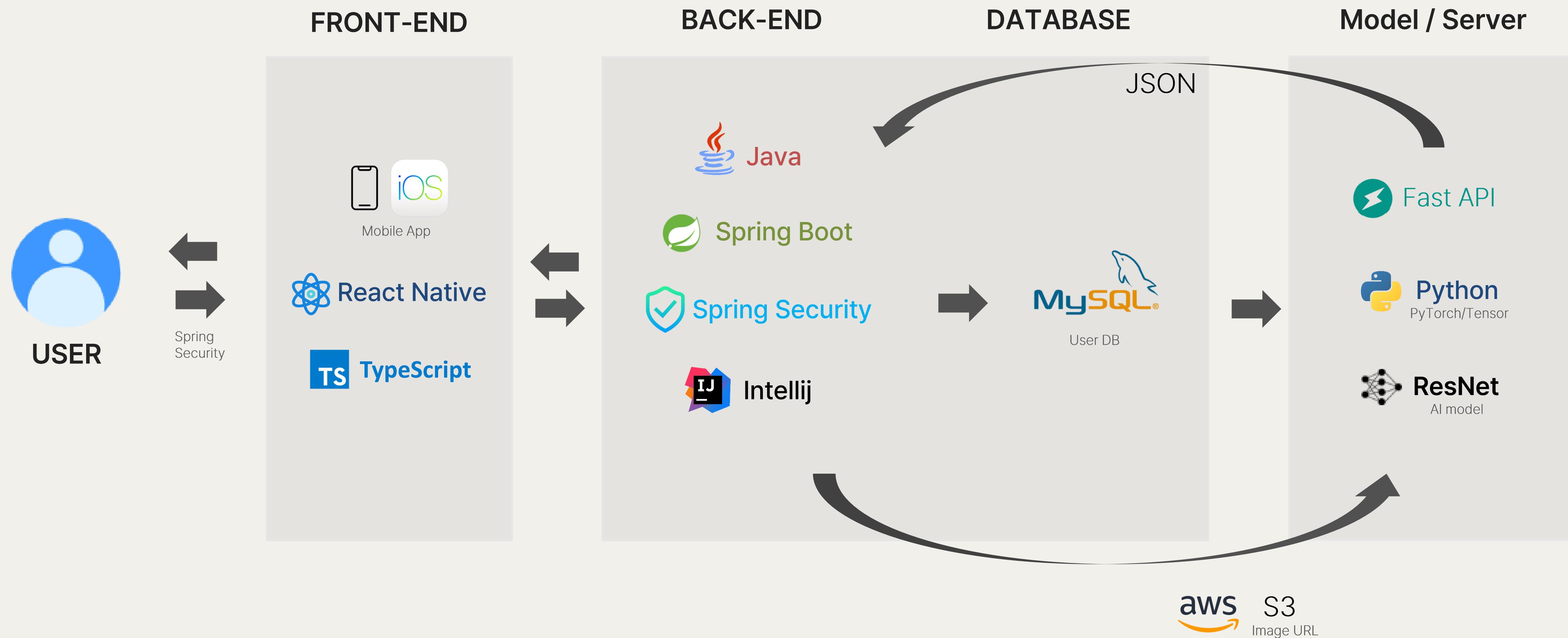


https://www.rootsanalysis.com/reports/digital-age-skincare-market.html?utm_source=chatgpt.com

화장품·뷰티 산업에서의 4차 산업혁명 도입은 비교적 최근의 현상이지만, AI와 ICT 등 첨단 기술의 진보에 힘입어 산업 전반이 초개인화가 가능한 데이터 중심 시장으로 재편되고 있다. 이에 따라 글로벌 화장품·뷰티 브랜드들은 경쟁력 강화를 위해 빅데이터와 인공지능 기반의 사업 전략을 본격적으로 수립·적용해 나가고 있다.

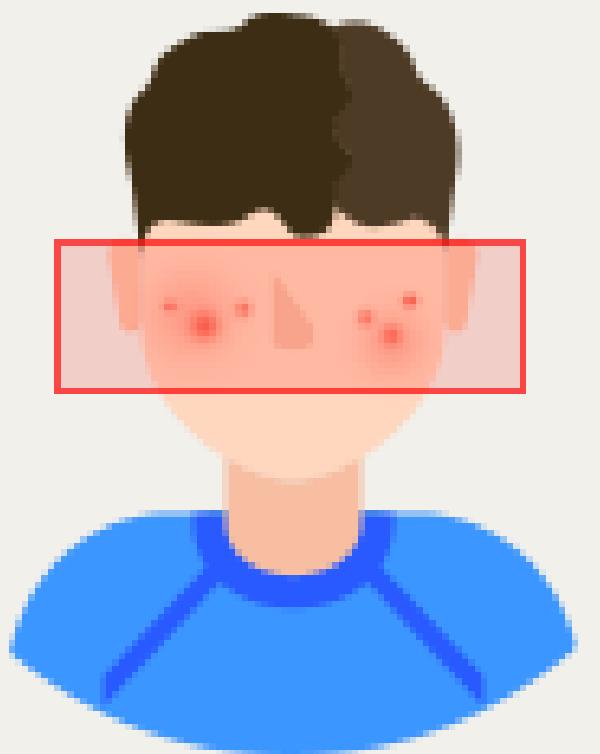
글로벌 화장품·뷰티 AI 시장은 빠른 속도로 성장하고 있다. 2022년 26억 8000만 달러(약 3조 5580억 원) 규모였던 시장은 2023년 32억 7000만 달러(약 4조 3410억 원)로 성장했을 것으로 예상되며, 2024년부터 2031년까지 연평균 20.1%의 고성장이 지속될 것으로 전망된다.

프로젝트 구성도



모듈 / 데이터 분석

데이터 수집



헬스케어 건강서비스

한국인 피부상태 측정 데이터

분야 | 영상 이미지
유형 | 텍스트, 이미지



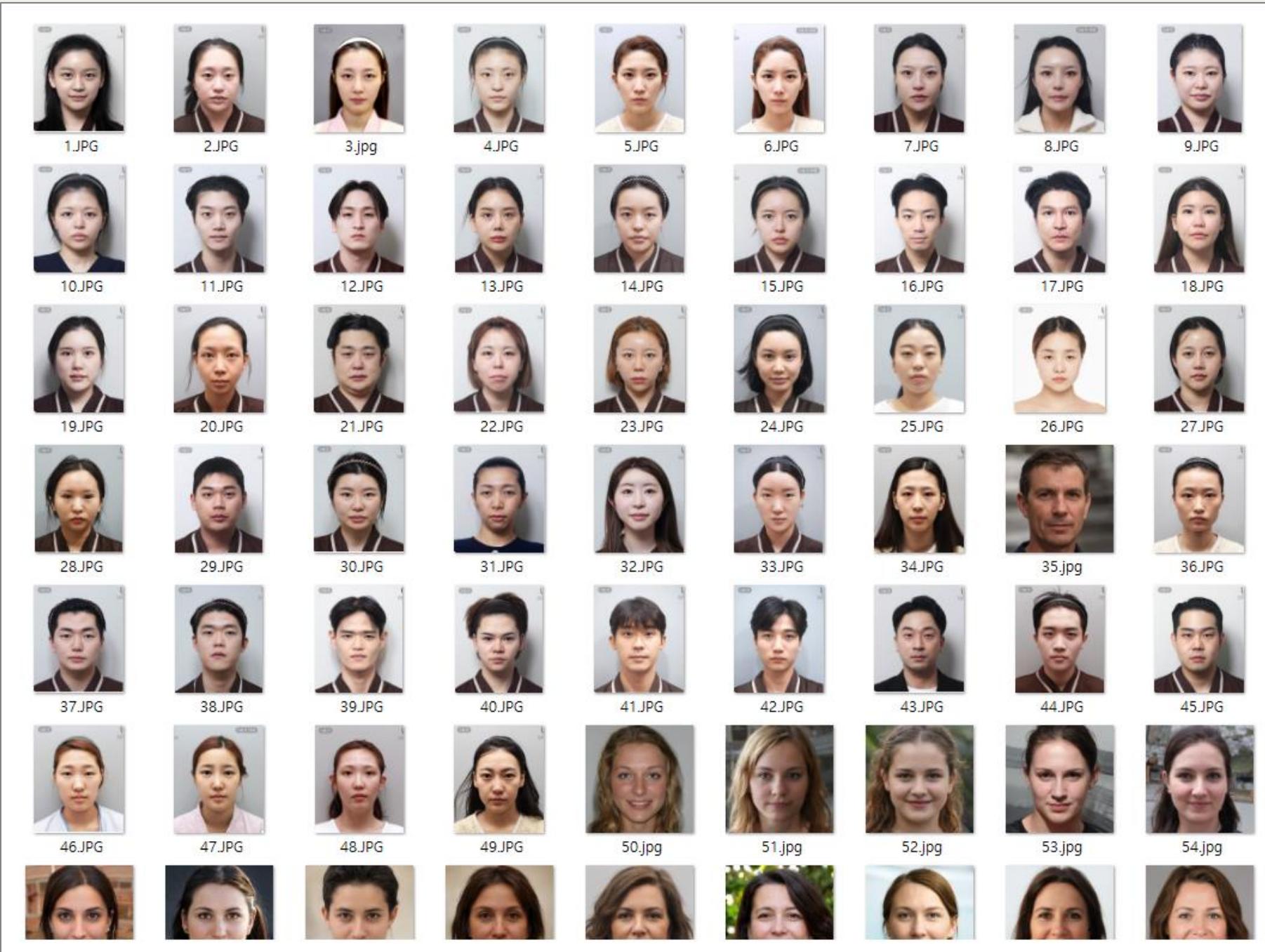
- 데이터 형식 **jpg, csv, json**
- 고해상도 다각도 안면이미지
13,936건
- 피부상태 측정 데이터
84,688건
- 메타데이터
6,432건



출처

[https://aihub.or.kr/aihubdata/data/view
.do?pageIndex=1&dataSetSn=71645](https://aihub.or.kr/aihubdata/data/view.do?pageIndex=1&dataSetSn=71645)

얼굴 이미지 150장



제품 크롤링

```
public HomeRecommendationSection recommend(ConditionType type, int score) { 5개 사용 위치 ☰ Seo
    // 1 DB 기반 추천
    List<CosmeticDto> dbCosmetics =
        cosmeticRepository.findTop5ByCategory(type).List<Cosmetic>
            .stream() Stream<Cosmetic>
            .map(CosmeticDto::from) Stream<CosmeticDto>
            .toList();
    // 2 네이버 실시간 추천 (🔥 수정 포인트)
    List<CosmeticDto> realtimeCosmetics =
        naverClient.search(keyword: type.getKoreanName() + " 화장품") NaverSearchResponse
            .getItems() List<NaverShoppingItem>
            .stream() Stream<NaverShoppingItem>
            .map(this::fromNaver) Stream<CosmeticDto>
            .toList();
    // 3 병합 (DB 우선 → 네이버 보강)
    List<CosmeticDto> merged =
        Stream.concat(dbCosmetics.stream(), realtimeCosmetics.stream())
            .distinct()
            .limit(maxSize: 10)
            .toList();
    return new HomeRecommendationSection(type, score, merged);
}
```

탄력

집중 케어가 필요해요

톤28 해남404 펩타시
카 새벽크림 50ml, 1개
톤28

설화수 탄력 크림 EX
75ml, 1개
설화수

수분

집중 케어가 필요해요

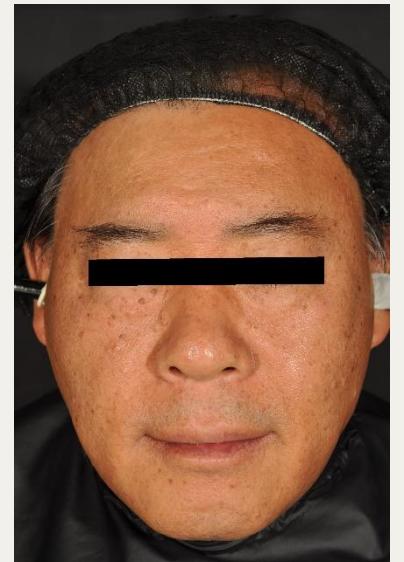
리로슈포제 시카플라스
트 밤 B5+ 100ml,...
리로슈포제

에스트라 아토베리어
365 크림 80ml(신...
에스트라

모델 정의서

학습 이미지

(기기, 각도별 13장의 사진)



>>
학습

라벨링 데이터 (전문장비를 통한 79개의 수치값)

subject_no	MOISTURE_FOREHEAD	MOISTURE_RIGHTCHEEK	MOISTURE_LEFTCHEEK	ELASTICITY_CHIN_R0	ELASTICITY_CHIN_R1	ELASTICITY_CHIN_R2	ELASTICITY_CHIN_R3	ELASTICITY_CHIN_R4	ELASTICITY_CHIN_R5	ELASTICITY_CHIN_R6	ELASTICITY_CHIN_R7	ELASTICITY_CHIN_R8	ELASTICITY_CHIN_R9
1	53	76.33	71.67	78.67	0.204	0.083	0.5931						

ooo

다중 회귀 분석 모델

ResNet18

Resizing : 224 x 224

Epoch = 10+30

Batch = 4

Optimizer : Adam

전이 학습 방식

Backbone LR = 0.001

UnFreeze LR = 0.0001

>>
수치 정규화 출력

JSON 파일

```
[{"image": "sample.jpg", "MOISTURE_FOREHEAD": 59, "MOISTURE_RIGHTCHEEK": 66, "MOISTURE_LEFTCHEEK": 65, "MOISTURE_CHIN": 67, "ELASTICITY_CHIN_R0": 19, "ELASTICITY_CHIN_R1": 27, "ELASTICITY_CHIN_R2": 48, "ELASTICITY_CHIN_R3": 20, "ELASTICITY_CHIN_R4": 46, "ELASTICITY_CHIN_R5": 50, "ELASTICITY_CHIN_R6": 46, "ELASTICITY_CHIN_R7": 46, "ELASTICITY_CHIN_R8": 17, "ELASTICITY_CHIN_R9": 49},
```

>>
서버 전송

모델 선별 과정

SimpleCNN

- 가장 단순하고 간단한 CNN 구조 모델로 시작
- Conv + Pool 반복, 빠른 실험과 기본 성능 확인
- 산출된 모델(.pt) 약 120mb

MobileNet

- 모바일 개발 환경에 맞춘 모델을 탐색
- 적은 파라미터수로 경량화에 특화되어있음
- 산출된 모델(.pt) 약 9mb

Resnet18

- 성능과 경량화를 모두 잡기 위한 선택
- Residual Block을 사용하여 SimpleCNN보다 더욱 안정적
- Mobilenet보다는 더 많은 파라미터로 경량화 일부 충족
- 산출된 모델(.pt) 약 40mb

SimpleCNN Validation 학습

모델 학습 과정

환경 설정	MAE	R2	소요시간	분석
Batch_size : 4 Img resize : 256 X 256 Epoch : 20	0.0810	0.6240	22m 28.1s	가장 최초 배경 테스트
Batch_size : 8 Img resize : 256 X 256 Epoch : 30	0.1060	0.3691	27m 48.8s	배치사이즈를 조금 상향하였으나 R2값이 급격히 저조
Batch_size : 16 Img resize : 256 X 256 Epoch : 20	0.1206	0.2068	17m 43.0s	에폭과 배치 사이즈를 줄이니 시간도 줄었지만 성능이 더욱 저조
Batch_size : 32 Img resize : 256 X 256 Epoch : 30	0.1181	0.2473	16m 56.2s	배치 사이즈를 더욱 상향하였음
Batch_size : 4 Img resize : 256 X 256 Epoch : 30 인원별 평균 Pooling	0.1748	-0.6584	17m 43.0s	매칭 방식을 변경하였으나 성능이 좋지 않음
Batch_size : 4 Img resize : 256 X 256 Epoch : 30	0.0678	0.7293	27m 48.8s	같은 환경에서도 다른 지표를 볼 수 있음

*성능과 시간을 고려하여 가장 적합한 환경을 찾기 위한 초반 테스트

- 시간은 조금 더 걸리나 배치 사이즈가 낮을수록 MAE와 R2의 성능이 향상됨
- 같은 환경에 재학습 R2 0.62 -> 0.73 으로 변화 하였음
- 이미지와 라벨링 1:1매칭 방식에서 인원별(13)장 평균 Pooling데이터에 라벨링을 매칭하는 방식으로는 성능이 나오지 않음
- 결론 : 1:1 매칭방법과 배치사이즈는 4로 수행

SimpleCNN 모델 Blur 처리한 이미지 예측

```
# GaussianBlur (흐리게 처리)

image_transform = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.GaussianBlur(kernel_size=(3,3), sigma=(0.1, 0.8)), # 초점이 살짝 나간정도의 흐림
    transforms.ToTensor()
])

GB_tf_images, GB_tf_labels = preview_transform(image_transform)

X1 = GB_tf_images
Y1 = GB_tf_labels
print(X1.size(), Y1.size())
```



[INFO] Loaded 1391 images from data/Validation/VS
Blur 영향 평균 절대 차이: 0.000859

- 원본 이미지 vs 블러 이미지를 넣었을 때 모델 예측값이 얼마나 달라졌는지를 측정
- 0.000859 → 아주 작은 변화량으로 거의 안 변했다는 뜻
- $0.000859 / 0.1 =$ 약 0.86% 크기로
- 전체 범위 기준 1%도 안 되는 변화
- 모델이 표현할 수 있는 변화량 중에서 블러가 만든 변화는 거의 티도 안 날만큼 작다.
- 이미지가 조금 흐려져도 결과가 안정적이다.

SimpleCNN Training 학습(1)

모델 학습 과정

환경 설정	MAE	R2	소요시간	분석
Batch_size: 4 Img resize: 256 X 256 Epoch: 5	0.0886	0.5662	44m 31.5s	적은 에폭으로 테스트 학습 수행
Batch_size: 4 Img resize: 256 X 256 Epoch: 20	0.0850	0.5958	170m 53.4s	위와 동일한 환경으로 에폭만 증가하였으나 성능이 저조
Batch_size: 4 Img resize : 512 X 512 Epoch: 40	0.0640	0.7566	397m 45.7s	에폭은 더욱 늘리고 이미지 특징을 잡기 위해 리사이징 크기 증가



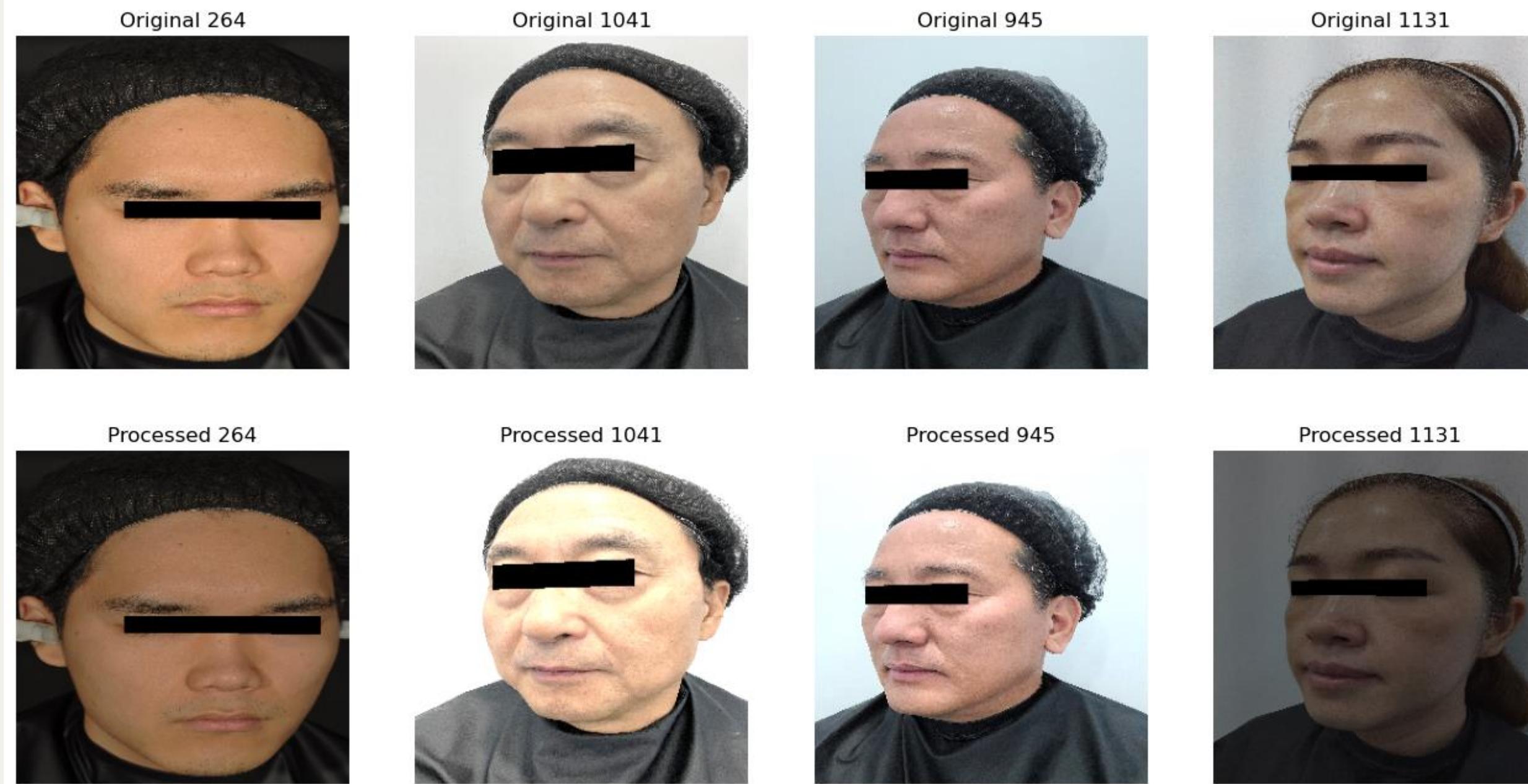
✓ 397m 45.7s

```

Epoch 1, Loss: 0.1086, MAE: 0.1248, R2: -2.5862
Epoch 2, Loss: 0.0200, MAE: 0.1100, R2: 0.3405
Epoch 3, Loss: 0.0171, MAE: 0.1015, R2: 0.4346
Epoch 4, Loss: 0.0136, MAE: 0.0899, R2: 0.5523
Epoch 5, Loss: 0.0112, MAE: 0.0815, R2: 0.6297
Epoch 6, Loss: 0.0101, MAE: 0.0771, R2: 0.6656
Epoch 7, Loss: 0.0096, MAE: 0.0749, R2: 0.6836
Epoch 8, Loss: 0.0092, MAE: 0.0731, R2: 0.6959
Epoch 9, Loss: 0.0089, MAE: 0.0717, R2: 0.7066
Epoch 10, Loss: 0.0087, MAE: 0.0708, R2: 0.7128
Epoch 11, Loss: 0.0085, MAE: 0.0698, R2: 0.7198
Epoch 12, Loss: 0.0084, MAE: 0.0692, R2: 0.7237
Epoch 13, Loss: 0.0083, MAE: 0.0686, R2: 0.7274
Epoch 14, Loss: 0.0082, MAE: 0.0680, R2: 0.7309
Epoch 15, Loss: 0.0081, MAE: 0.0676, R2: 0.7339
Epoch 16, Loss: 0.0079, MAE: 0.0669, R2: 0.7377
Epoch 17, Loss: 0.0079, MAE: 0.0667, R2: 0.7396
Epoch 18, Loss: 0.0078, MAE: 0.0664, R2: 0.7413
Epoch 19, Loss: 0.0078, MAE: 0.0661, R2: 0.7429
Epoch 20, Loss: 0.0077, MAE: 0.0658, R2: 0.7450
Epoch 21, Loss: 0.0077, MAE: 0.0657, R2: 0.7456
Epoch 22, Loss: 0.0077, MAE: 0.0654, R2: 0.7472
Epoch 23, Loss: 0.0076, MAE: 0.0653, R2: 0.7478
Epoch 24, Loss: 0.0076, MAE: 0.0651, R2: 0.7491
Epoch 25, Loss: 0.0076, MAE: 0.0651, R2: 0.7491
...
Epoch 38, Loss: 0.0074, MAE: 0.0640, R2: 0.7559
Epoch 39, Loss: 0.0074, MAE: 0.0640, R2: 0.7559
Epoch 40, Loss: 0.0074, MAE: 0.0640, R2: 0.7566
모델 저장 완료: skin_model12.pt

```

밝기 변화로 예측, 정확도 평가



- MAE = 0.066, R² = 0.737로
예측 오차가 크지 않고, 밝기 변화 조건에서도 비교적 높은 설명력 유지
- 모델이 조명 변화에 대해 일정 수준의 안정성을 가짐

```
def evaluate(model, loader):
    model.eval()

    all_preds = []
    all_labels = []

    with torch.no_grad():
        for imgs, labels in loader:
            imgs = imgs.to(device)

            preds = model(imgs).cpu()
            all_preds.append(preds)
            all_labels.append(labels)

    all_preds = torch.cat(all_preds)
    all_labels = torch.cat(all_labels)

    mae = torch.mean(torch.abs(all_preds - all_labels)).item()
    ss_res = torch.sum((all_labels - all_preds)**2).item()
    ss_tot = torch.sum((all_labels - torch.mean(all_labels))**2).item()
    r2 = 1 - ss_res/ss_tot

    return mae, r2

mae, r2 = evaluate(model, train_loader)
print("Test MAE:", mae)
print("Test R2:", r2)
```

Test MAE: 0.06645150482654572
Test R2: 0.7378605033365131

이미지 로테이션으로 예측, 정확도 평가

```
# 이미지의 각도 회전 RandomRotation (degrees=10)

image_transform = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.RandomRotation(degrees=10),
    transforms.ToTensor()
])
```



```
correct = 0
total = 0

with torch.no_grad():
    for batch in transform_loader:
        img = batch[0].to(device)
        label = batch[1].to(device)

        outputs = model(img)
        _, predicted = torch.max(outputs, 1)

        total += label.size(0)
        correct += (predicted == label).sum().item()

accuracy = 100 * correct / total
print(f"회전된 이미지 정확도: {accuracy:.2f}%")
```

회전된 이미지 정확도: 1.94%

- 1.94%라는 건 거의 무작위 수준이라는 뜻
- 모델이 회전 데이터로 학습된 적이 없음
- CNN은 기본적으로 회전에 매우 약함

이미지 로테이션으로 예측, 정확도 평가

주요 발견사항

- 세로(열)를 기준으로 보면 예측이 거의 특정 세 클래스에만 집중됨

Predicted=3,4,5

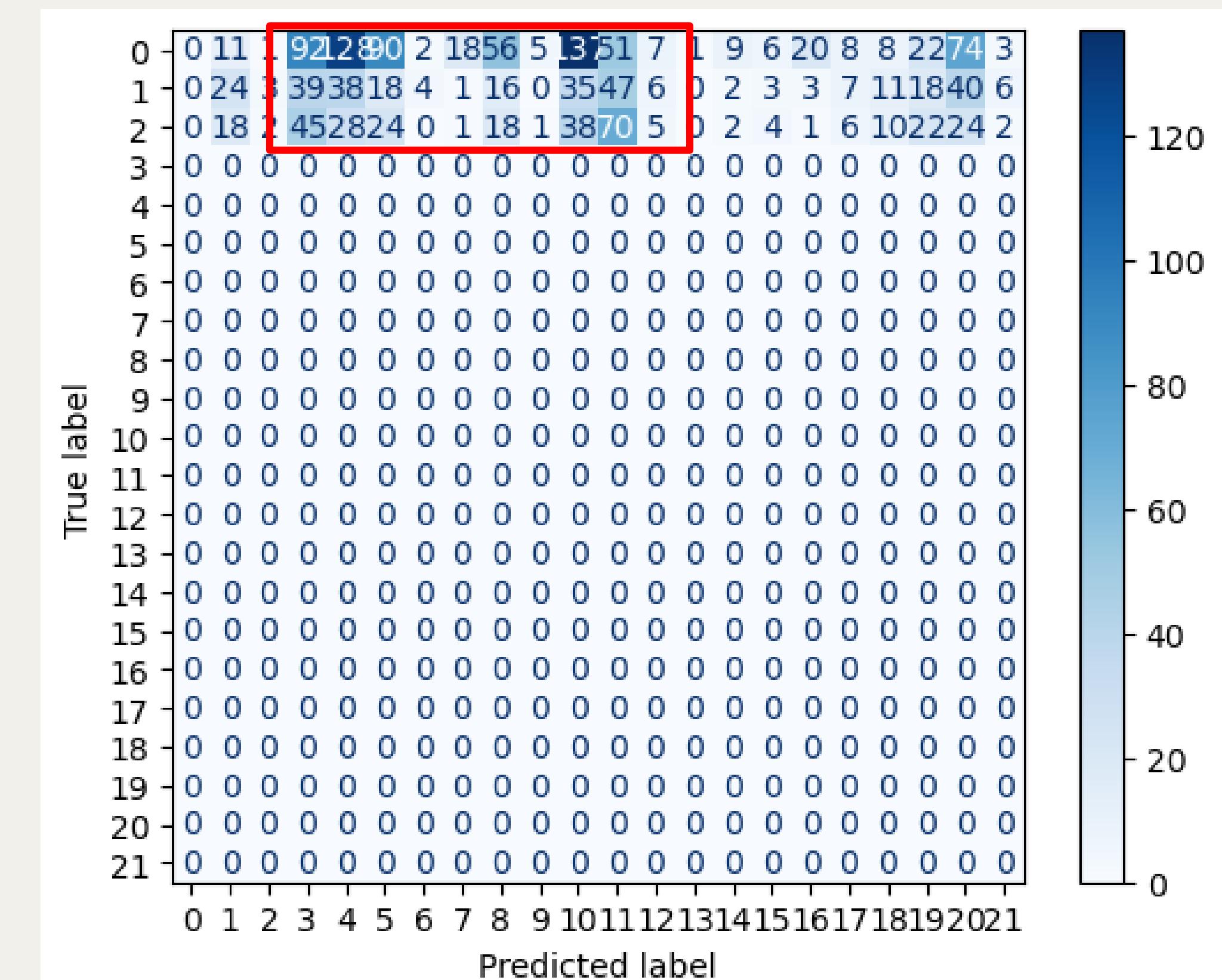
- 실제 라벨이 무엇이든 대부분 3,4,5번 클래스로 예측됨
 - 모델이 중간 범위 값으로만 수렴하는 경향

상태

- 모델이 다양한 클래스를 구분하지 못하고, 입력을 소수 클래스(3,4,5,10)로 몰아넣는 상태.
 - “예측 다양성 붕괴(Class Collapse)”
 - 형태가 바뀌어서 CNN이 원래 클래스의 특징을 잡지 못함.

이우

- 데이터 분포가 바뀌어서
 - 회전시키면 이미지의 질감/패턴/위치가 바뀜 → CNN이 못 알아볼 수 있음



SimpleCNN Training 학습(2)

모델 학습 과정

환경 설정	MAE	R2	소요시간	분석
Batch_size : 4 Img resize : 256 X 256 Epoch : 35	0.0541	0.8234	321m 21.9s	이전 학습 중 모델보다 성능이 낮아 학습 중단
Batch_size : 4 Img resize : 256 X 256 Epoch : 50	0.0403	0.9002	399m 21.5s	최적의 환경으로 반복 수행 결과 지표가 가장 좋은 모델

* 배치 사이즈 4의 환경으로 더 나은 지표를 얻기 위한 테스트

- 학습을 수행할 때마다 랜덤한 가중치를 부여하기에 이후 진행되는 에폭에 결과가 다르다는 걸 재확인함
- 같은 환경에 재학습 R2 0.82 -> 0.90 으로 변화 하였음
- 결론 : SimpleCNN 모델 R2 0.9002 모델이 완성



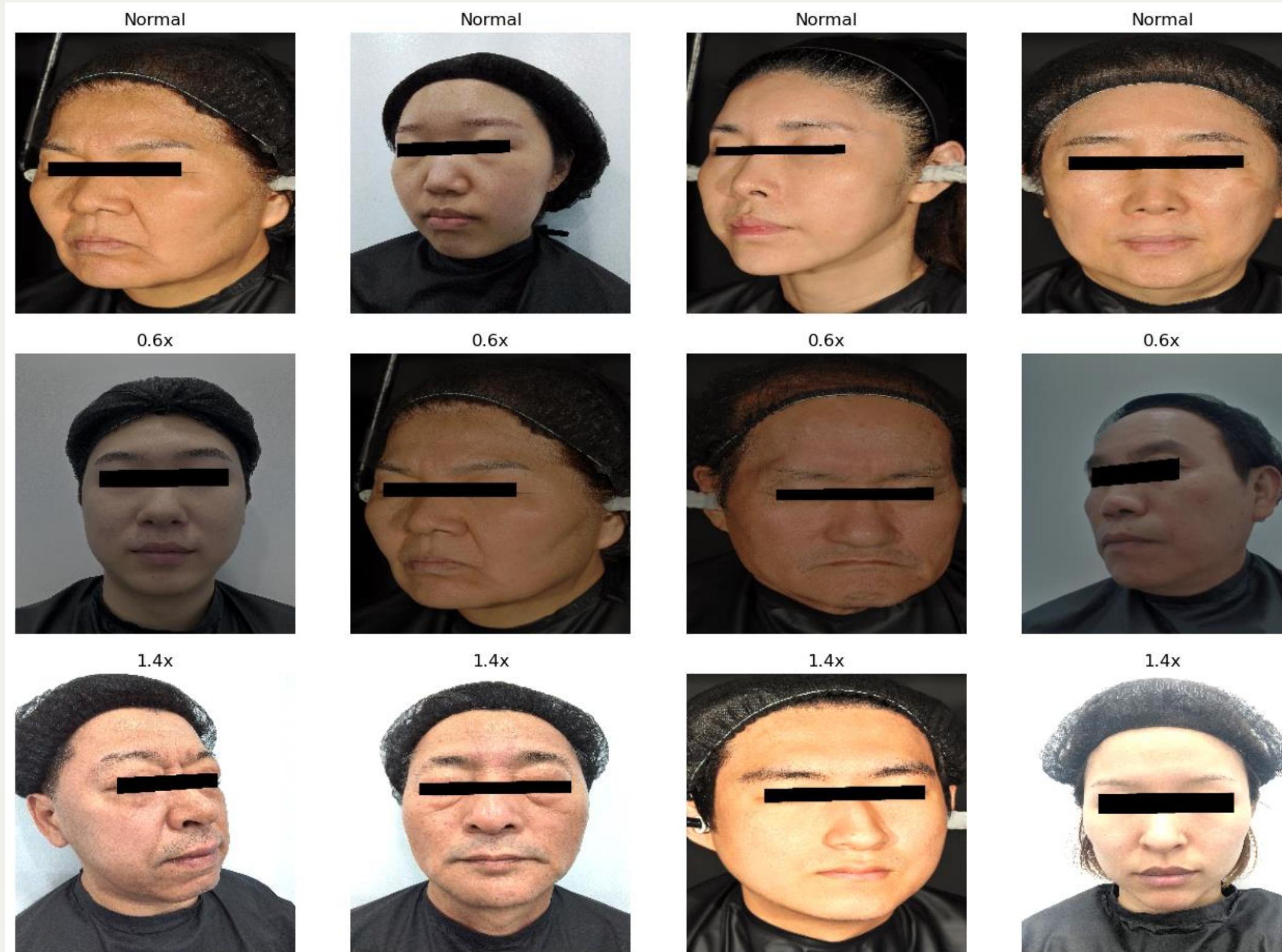
```

✓ 399m 21.5s

Epoch 1, Loss: 0.0215, MAE: 0.1144, R2: 0.2896
Epoch 2, Loss: 0.0194, MAE: 0.1086, R2: 0.3597
Epoch 3, Loss: 0.0179, MAE: 0.1040, R2: 0.4098
Epoch 4, Loss: 0.0158, MAE: 0.0975, R2: 0.4796
Epoch 5, Loss: 0.0132, MAE: 0.0891, R2: 0.5633
Epoch 6, Loss: 0.0109, MAE: 0.0807, R2: 0.6399
Epoch 7, Loss: 0.0092, MAE: 0.0738, R2: 0.6960
Epoch 8, Loss: 0.0080, MAE: 0.0685, R2: 0.7358
Epoch 9, Loss: 0.0072, MAE: 0.0647, R2: 0.7635
Epoch 10, Loss: 0.0066, MAE: 0.0617, R2: 0.7837
Epoch 11, Loss: 0.0061, MAE: 0.0593, R2: 0.7993
Epoch 12, Loss: 0.0058, MAE: 0.0575, R2: 0.8102
Epoch 13, Loss: 0.0054, MAE: 0.0557, R2: 0.8209
Epoch 14, Loss: 0.0052, MAE: 0.0543, R2: 0.8292
Epoch 15, Loss: 0.0050, MAE: 0.0533, R2: 0.8352
Epoch 16, Loss: 0.0048, MAE: 0.0523, R2: 0.8405
Epoch 17, Loss: 0.0046, MAE: 0.0513, R2: 0.8465
Epoch 18, Loss: 0.0045, MAE: 0.0504, R2: 0.8510
Epoch 19, Loss: 0.0044, MAE: 0.0498, R2: 0.8545
Epoch 20, Loss: 0.0043, MAE: 0.0491, R2: 0.8579
Epoch 21, Loss: 0.0042, MAE: 0.0485, R2: 0.8613
Epoch 22, Loss: 0.0041, MAE: 0.0479, R2: 0.8645
Epoch 23, Loss: 0.0040, MAE: 0.0474, R2: 0.8669
Epoch 24, Loss: 0.0040, MAE: 0.0469, R2: 0.8692
Epoch 25, Loss: 0.0039, MAE: 0.0465, R2: 0.8711
...
Epoch 48, Loss: 0.0031, MAE: 0.0405, R2: 0.8991
Epoch 49, Loss: 0.0030, MAE: 0.0404, R2: 0.8997
Epoch 50, Loss: 0.0030, MAE: 0.0403, R2: 0.9002
모델 저장 완료: skin_model13.pt

```

밝기 변화로 예측, 정확도 평가



- Normal > Bright > Dark 순으로 성능 유지
- 밝기 변화가 있을 경우, 성능 저하는 발생하지만, 밝은 조건에서는 비교적 안정적인 예측을 유지

```

def evaluate(model, loader):
    all_preds, all_labels = [], []
    with torch.no_grad():
        for imgs, labels in loader:
            imgs = imgs.to(device)
            preds = model(imgs).cpu()
            all_preds.append(preds)
            all_labels.append(labels)
    all_preds = torch.cat(all_preds)
    all_labels = torch.cat(all_labels)
    mae = torch.mean(torch.abs(all_preds - all_labels)).item()
    r2 = 1 - torch.sum((all_labels - all_preds)**2).item() / torch.sum((all_labels - all_labels.mean())**2).item()
    return mae, r2

mae_n, r2_n = evaluate(model, loader_normal)
mae_d, r2_d = evaluate(model, loader_dark)
mae_b, r2_b = evaluate(model, loader_bright)

print("Normal :", mae_n, r2_n)
print("Dark   :", mae_d, r2_d)
print("Bright :", mae_b, r2_b)

```

Normal : 0.039892490953207016 0.8986037093830919
 Dark : 0.08897779136896133 0.564716219082178
 Bright : 0.06579067558050156 0.7569003250833684

이미지 Blur 효과 적용 후 예측, 정확도 평가

```
# 이미지 데이터 BLUR

image_transform = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.GaussianBlur(kernel_size=5, sigma=(1.0, 2.0)),
    transforms.ToTensor()
])
```



- blur 영향 MAE: 0.2154
- 블러를 넣으면 모델 출력이 평균적으로 0.026 정도 변한다
- MAE 0.02 ~ 0.05 중간 정도 변화 (적당히 견고)
- 블러가 들어가면 출력이 살짝 흔들리긴 하는데, 완전 무너지는 정도는 아니다
- 정밀도는 떨어지거나 출력이 완전히 틀어지진 않음

```
import torch.nn.functional as F
import torchvision.transforms as T

blur = T.GaussianBlur(kernel_size=5, sigma=(1, 2))

total_mae = 0
count = 0

with torch.no_grad():
    for img, _ in transforms_loader: # label은 안 받는다
        img = img.to(device)

        preds_clean = model(img)
        preds_blur = model(blur(img))

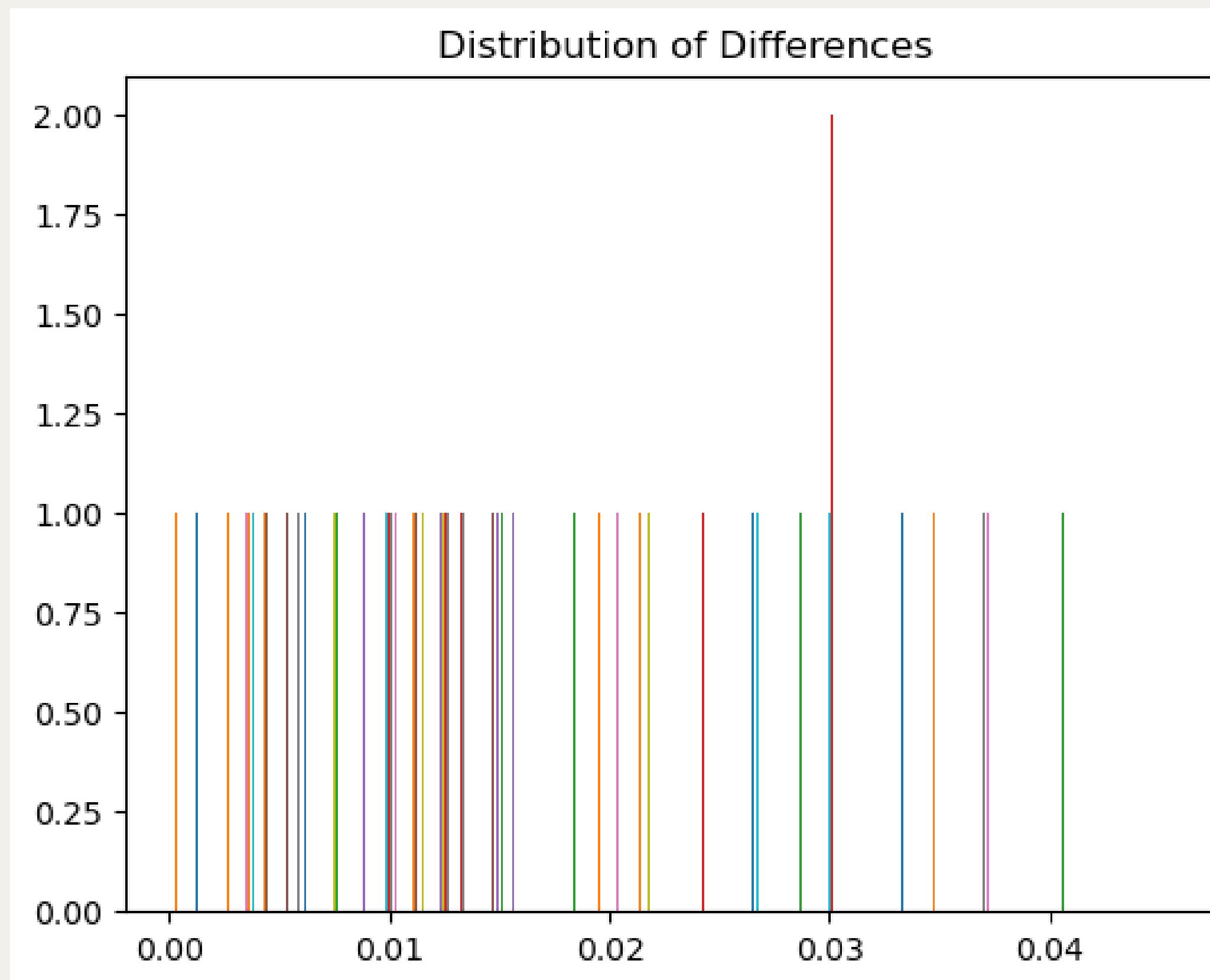
        mae = F.l1_loss(preds_blur, preds_clean, reduction='mean')

        total_mae += mae.item()
        count += 1

avg_mae = total_mae / count
print(f"블러 영향 MAE: {avg_mae:.4f}")
```

블러 영향 MAE: 0.0260

Distribution of Differences (차이 분포 그래프)

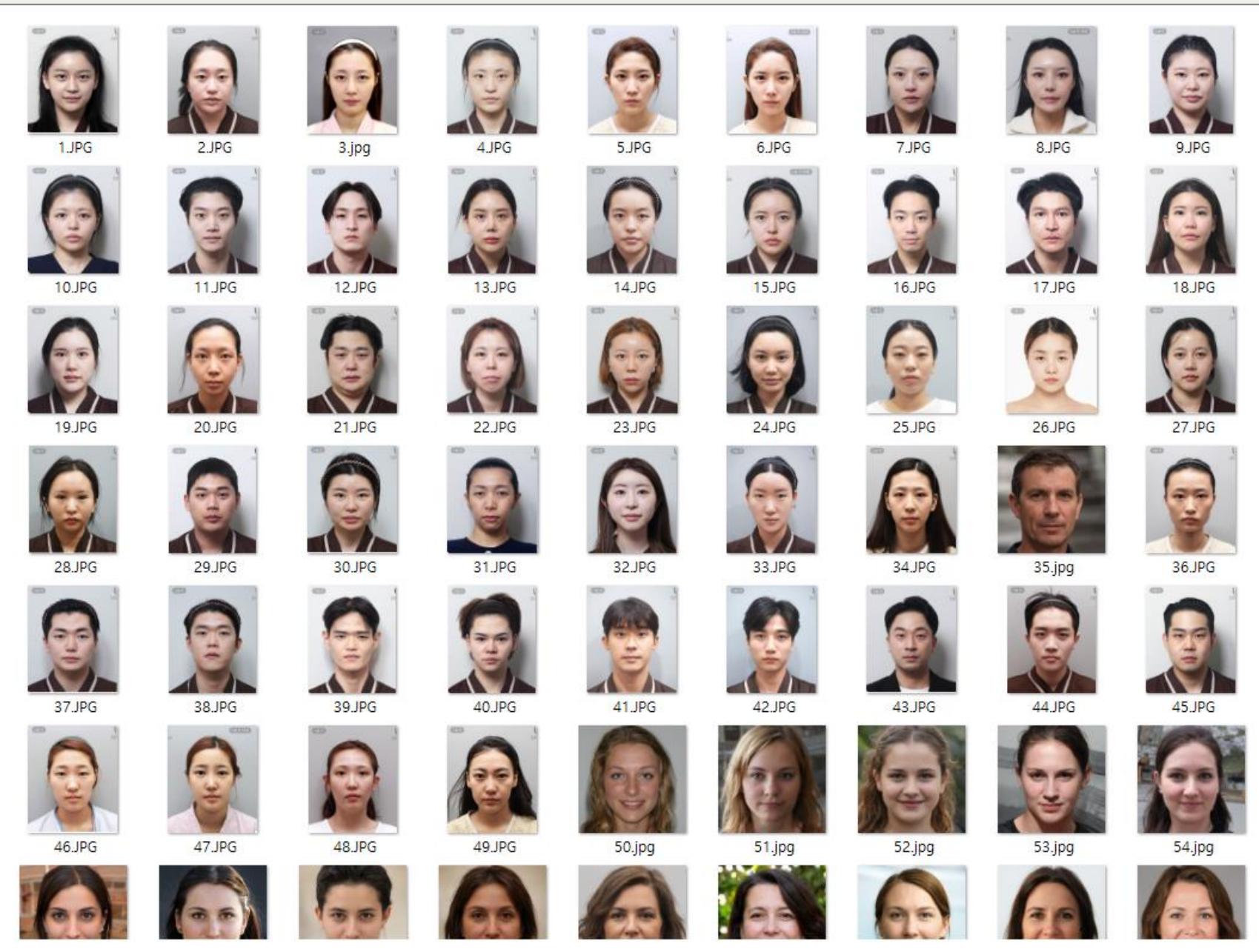


📌 그래프 설명

- 이 그래프는 블러를 적용하기 전(pred_clean) 과 블러 적용 후(pred_blur) 의 예측 차이를 보여줌.
- $|pred_clean - pred_blur|$ 를 각 출력 차원 별로 보여주는 분포로 79개 출력(피부 측정값) 중 어떤 값이 블러에 더 민감한지 확인할 수 있음.
- 블러 영향이 균일하지 않고, 특정 항목만 민감하게 반응함.
- 이는 회귀 기반 피부 측정 모델에서는 흔한 패턴임 (예: 색 기반 지표는 blur 영향 적고, 텍스처 기반 지표는 영향 큼)
- 주름/잔주름/미세 러프니스는 모두 고주파(high-frequency) 특징임.
- Gaussian Blur는 이미지에서 가 먼저 고주파 성분을 제거함.
- blur → 주름이 희미해짐 → 모델 예측이 크게 흔들림
- 모공, 주름, 탄력 R-계열, elasticity roughness, pore, wrinkle 등 텍스처 기반 요소가 blur에 매우 취약
- pigmentation, moisture 같은 저주파/색 기반 요소는 blur에 덜 민감

SimpleCNN 모델 측정 데이터

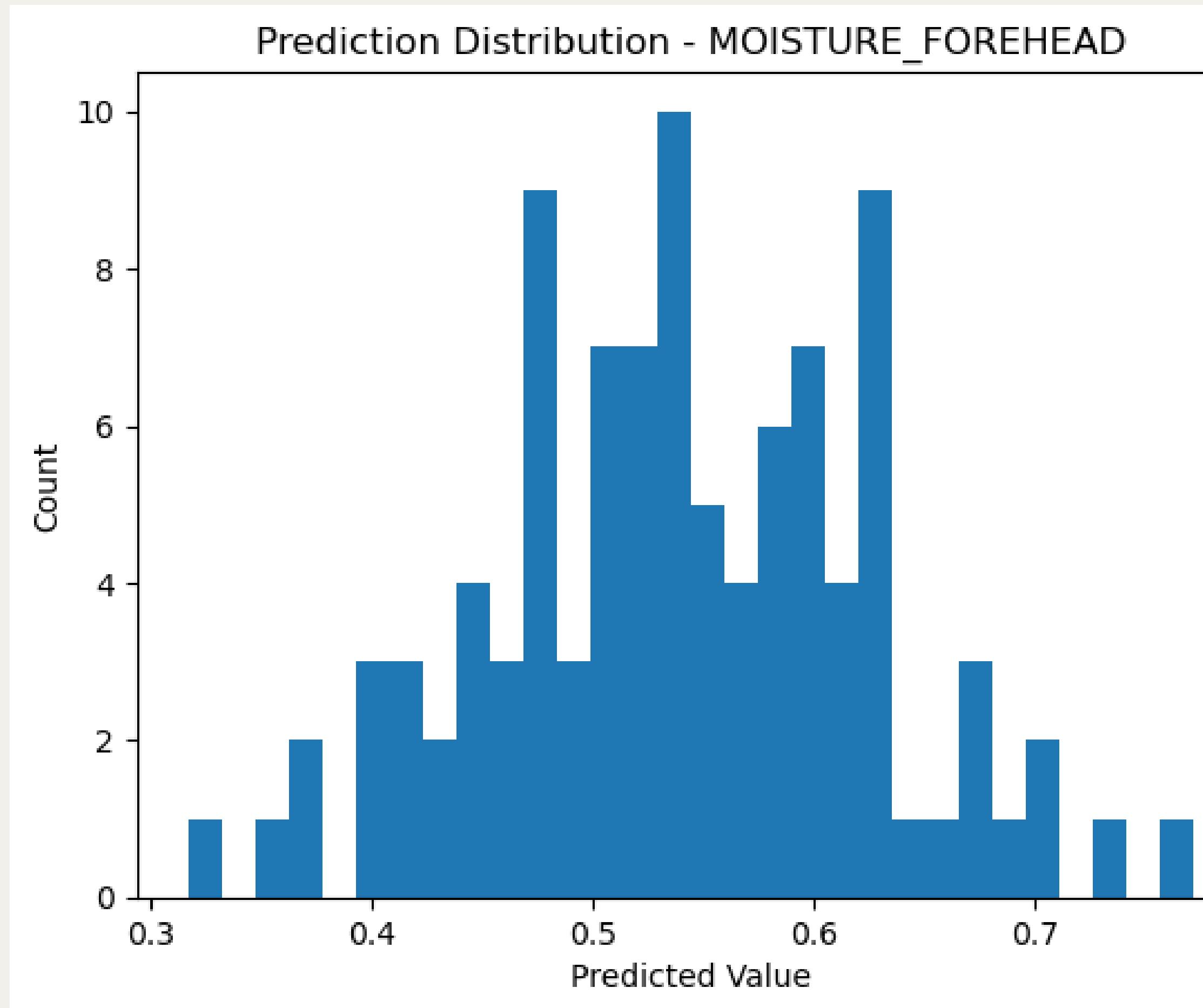
- 학습되지 않은 150명의 얼굴 정면 사진 수집
 - SimpleCNN 모델을 사용하여 측정 데이터 수집 및 분석



수집 목적

- 모델의 실제 환경 성능 검증
 - 다양한 피부 타입에 대한 예측 정확도 평가
 - 서비스 적용 가능성 확인

Prediction Distribution – MOISTURE_FOREHEAD (히스토그램)



➤ 그래프가 보여주는 것

- x축: 이마 수분 예측 값
- y축: 해당 값이 나온 이미지 개수

🔍 관찰 포인트

- 값이 약 0.45 ~ 0.6 구간에 가장 많이 몰려 있음
- 0.3 이하나 0.75 이상 같은 극단 값은 거의 없음
- 분포 형태가 대체로 종 모양

📝 해석

이마 수분 지표의 예측값은 특정 한 점에 고정되지 않고 합리적인 범위 내에서 분포하며, 극단적인 이상치는 거의 관찰되지 않는다.

☞ 의미:

모델 출력이 안정적이고 튀지 않는다.

MobileNet 및 ResNet 학습

MobileNet

환경 설정	MAE	R2	소요시간	분석
학습 데이터 : Validation Img resize : 224 X 224 Epoch : 5	0.1071	0.3699	6m 20.9s	MobileNet 기본 성능 테스트
학습 데이터 : Validation Img resize : 224 X 224 Epoch : 10+30	0.0489	0.8658	49m 27.8s	전이 학습방식을 적용하여 성능이 크게 향상되었음
학습 데이터 : Training Img resize : 224 X 224 Epoch : 10+27	0.0894	0.5603	513m23.3s	전체 데이터 학습 시 저조한 성능 지표
학습 데이터 : Validation Img resize : 224 X 224 Epoch : 10+57	0.0342	0.9349	65m 50.9s	Valid 데이터 재학습 후 상승한 지표가 나왔음

ResNet

환경 설정	MAE	R2	소요시간	분석
학습 데이터 : Validation Img resize : 256 X 256 Epoch : 10+30	0.0894	0.5609	54m 29.0s	Resnet 기본 성능 테스트
학습 데이터 : Training Img resize : 256 X 256 Epoch : 10+30	0.0238	0.9693	577m 35.5s	Training 학습시 가장 지표가 좋은 모델

* 성능을 더욱 상승시키거나 모델의 경량화를 위한 테스트

- SimpleCNN(120mb)과 비교하여 산출 모델의 크기가 MobileNet(9mb) / ResNet(40mb)으로 경량화에 좋은 모델
- MobileNet의 경우 전체 데이터 학습시 오히려 성능 지표가 저조함
- 두 모델 모두 전이 학습 방식을 통하여 성능을 더 개선하였음
- 결론 : ResNet으로 산출된 모델로 서비스를 구축하나 오버피팅이 의심됨

ResNet (train 파일만 학습) validation으로 검증

```
mae, r2, _, _ = evaluate_by_subject_mean(model, val_loader, device)
print("val images:", len(val_ds))
print(f"[VAL by subject] subjects={len(val_ids)} | MAE={mae:.4f} | R2={r2:.4f}")
```

✓ 42.4s

```
val images: 321
[VAL by subject] subjects=107 | MAE=0.1053 | R2=0.3952
```

➤ Validation 성능 결과 해석

검증 데이터 수 : 321장

MAE = 0.1082

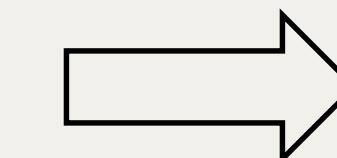
-> 실제 값 대비 평균적으로 약 0.11 정도의 오차로 예측

$R^2 = 0.3952$

-> 전체 변동성의 약 36%를 모델이 설명

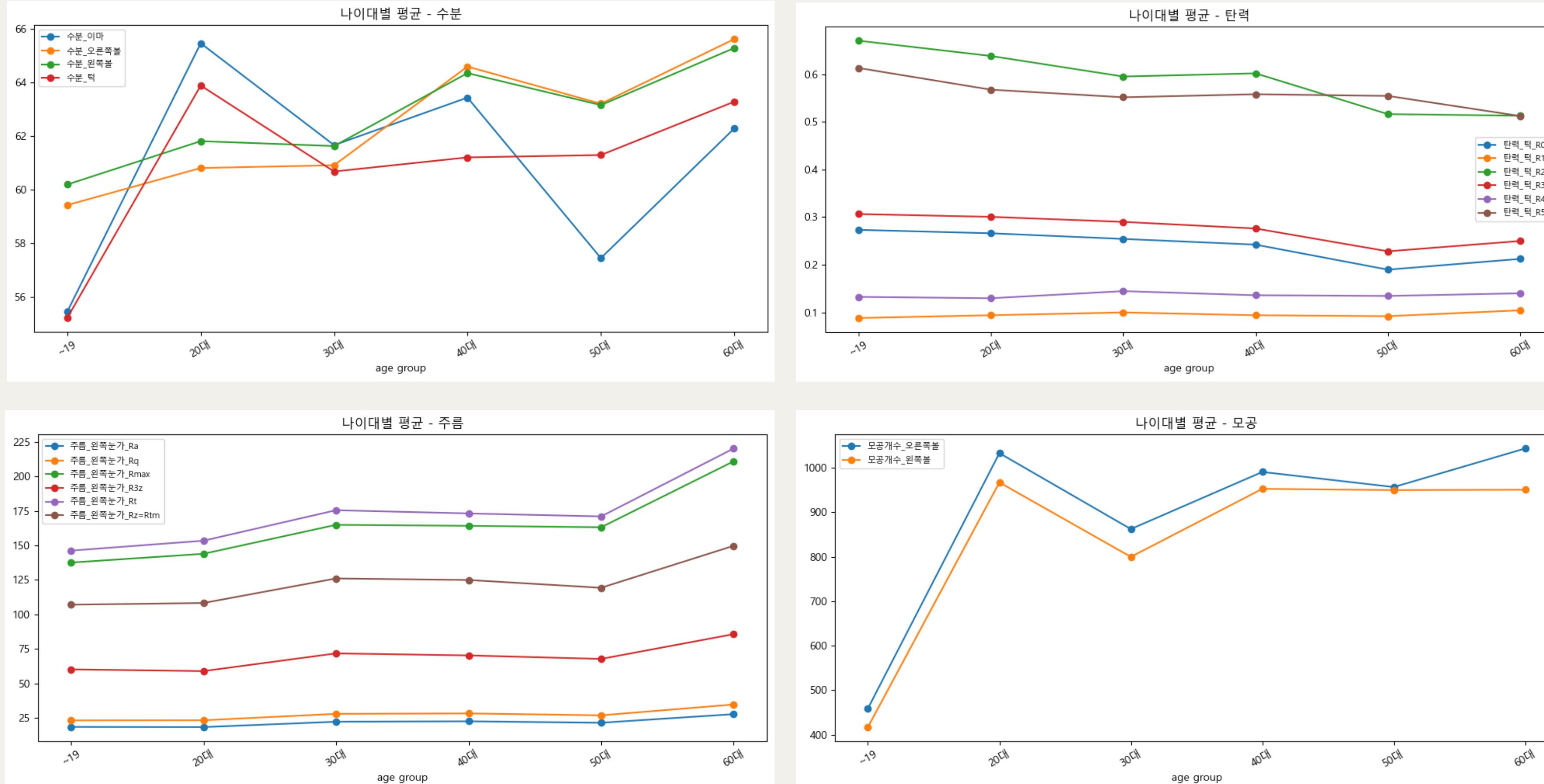
-> 완성 단계 모델이라기 보다는, 유의미한 패턴을 학습한

초기 성능 수준



피부 지표의 일부 패턴을 학습하는 데 성공했으나,
추가적인 데이터 및 모델 개선이 필요한 단계로 보임

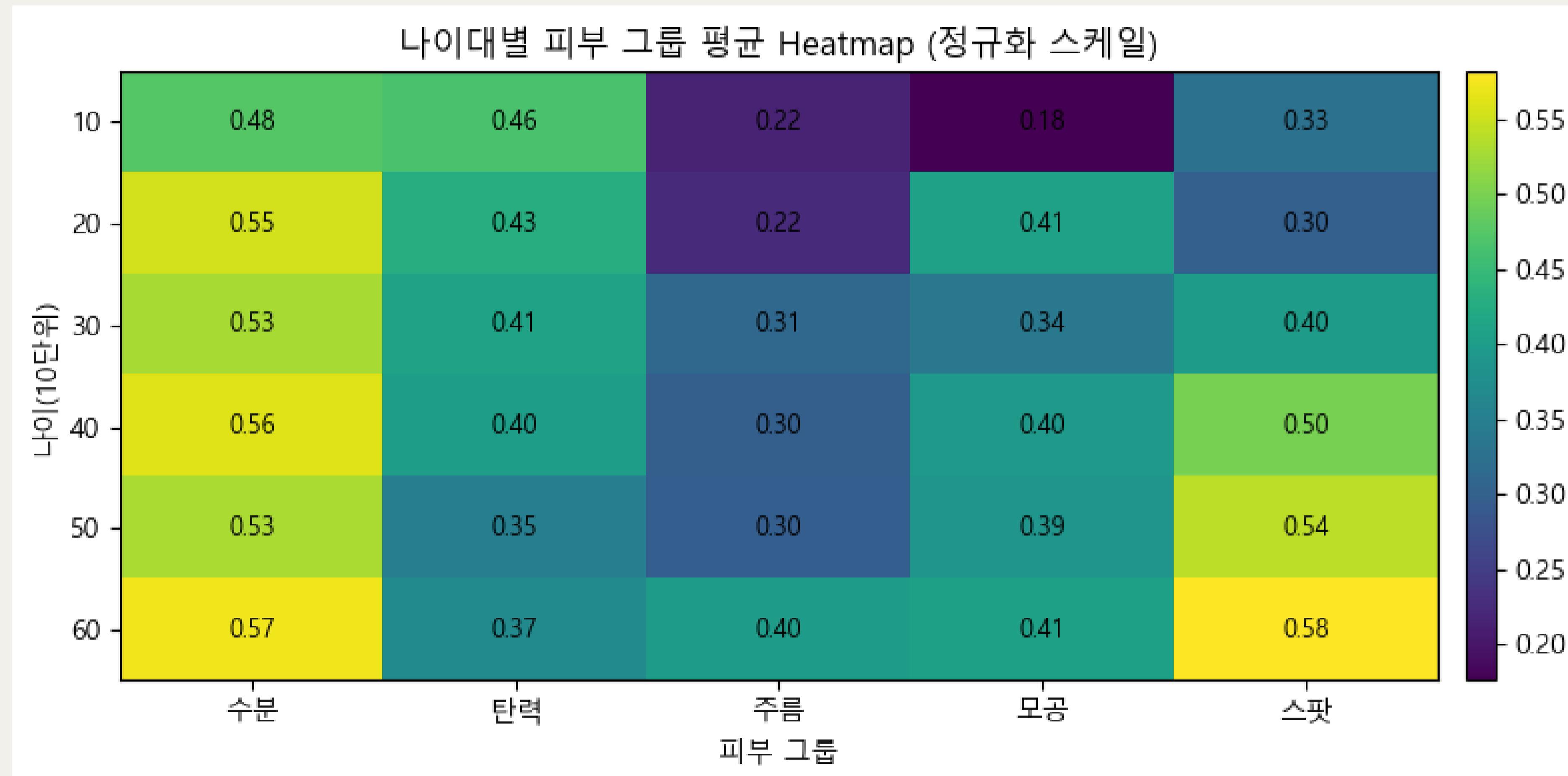
ResNet 검증 데이터 나이별 피부 지표 변화



- ☞ 그래프가 보여주는 것
- 연령 증가에 따라 수분 지표는 전반적으로 완만
 - 탄력 지표는 연령 증가에 따라 점진적으로 감소
 - 주름 관련 지표는 40대 이후 증가 폭이 커짐
 - 모공 개수는 연령대가 높아질수록 증가하는 경향

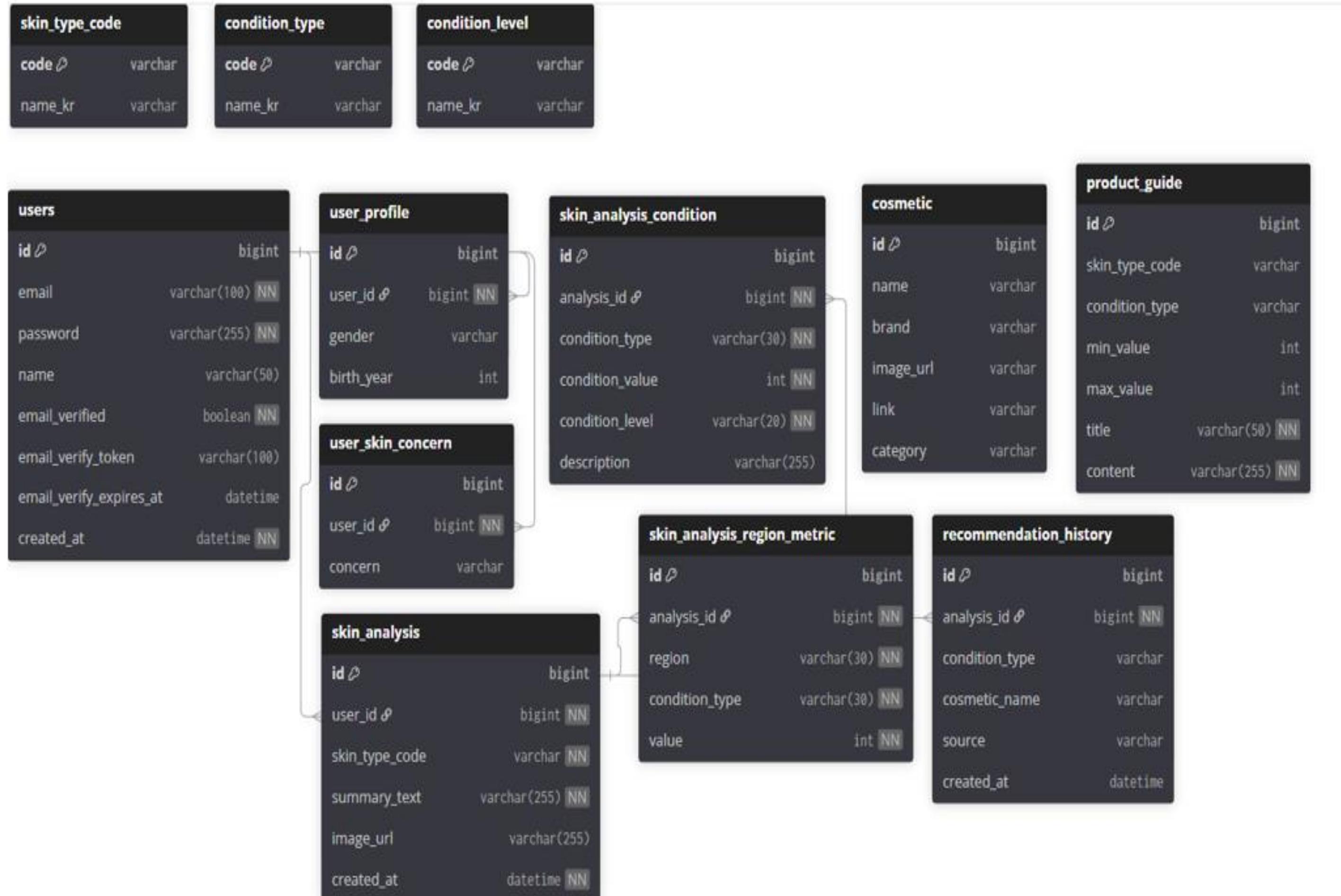
☞ 해석
나이별 평균 비교 결과,
수분·탄력·주름·모공 지표 모두 연령
증가에 따른 변화 패턴을 보였으며,
모델 예측 결과가 실제 피부 노화 경향과
일관됨을 확인하였다.

ResNet 검증 데이터 나이별 피부 Heatmap



❖ 정규화된 Heatmap에서도 나이 증가에 따라 주름·모공·스팟 지표의 상대적 값이 높아지는 패턴이 확인됨.

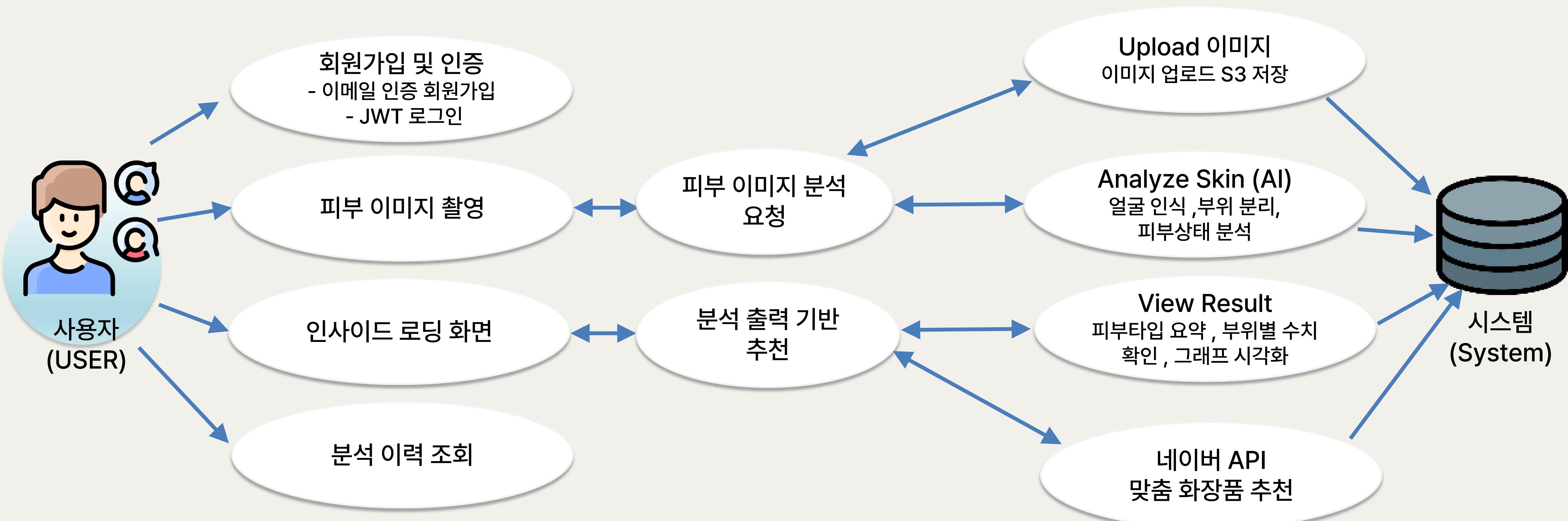
서비스 구현

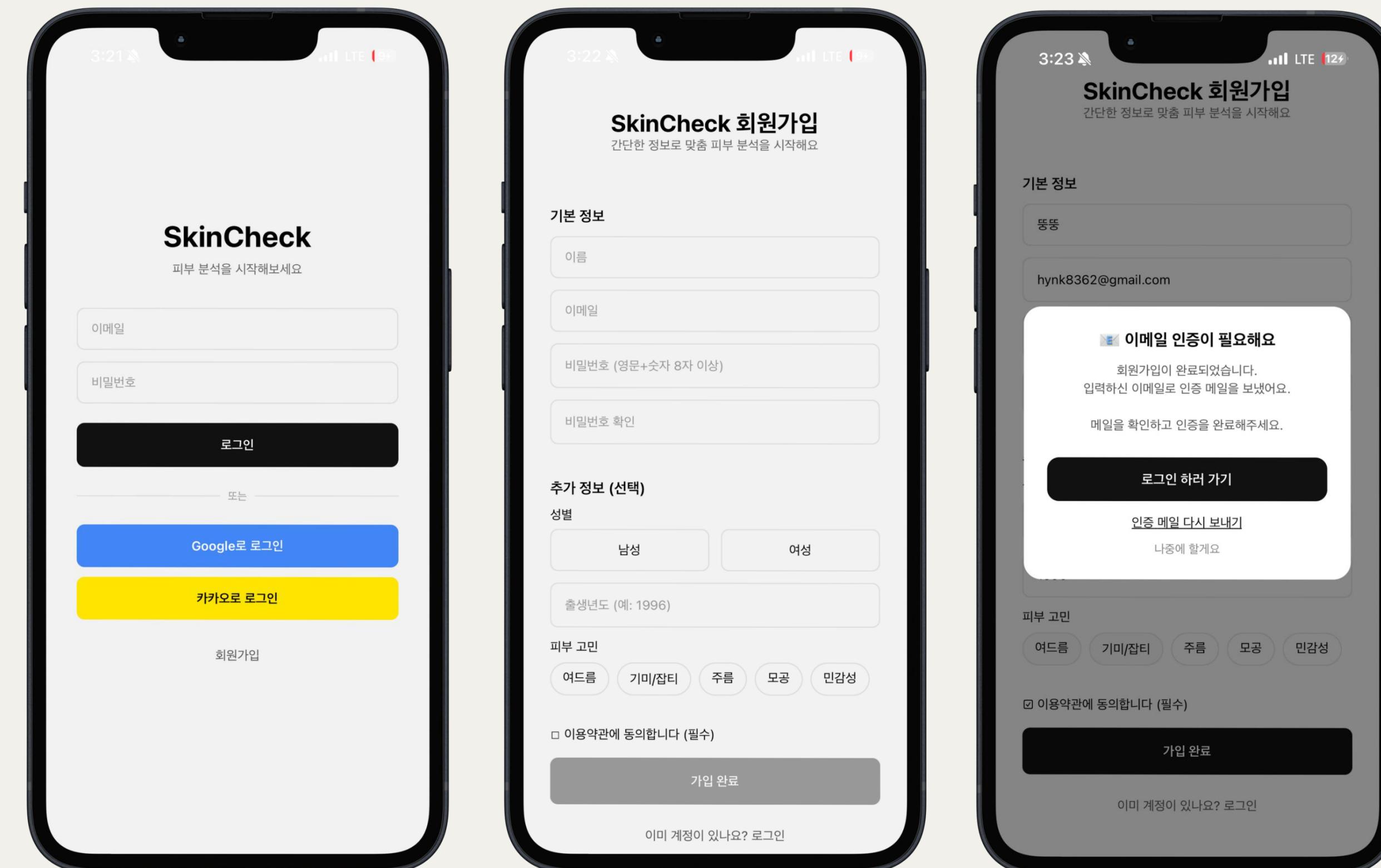


ERD 간편 설명

- ERD는 사용자를 중심으로 피부 분석과 추천 기능을 관리하도록 설계
- 각 피부 분석은 하나의 분석 정보와 여러 개의 세부 지표로 구성
- 분석 (SKIN_ANALYSIS)
분석지표(SKIN_ANALYSIS_CONDITION)
- AI 연동 로그와 추천 이력을 별도 관리해, 추적 가능성과 향후 기능 확장을 고려했습니다.

USE CASE





COMP TYPE 1

인증 (Auth) 화면 구성 - 구현 설명

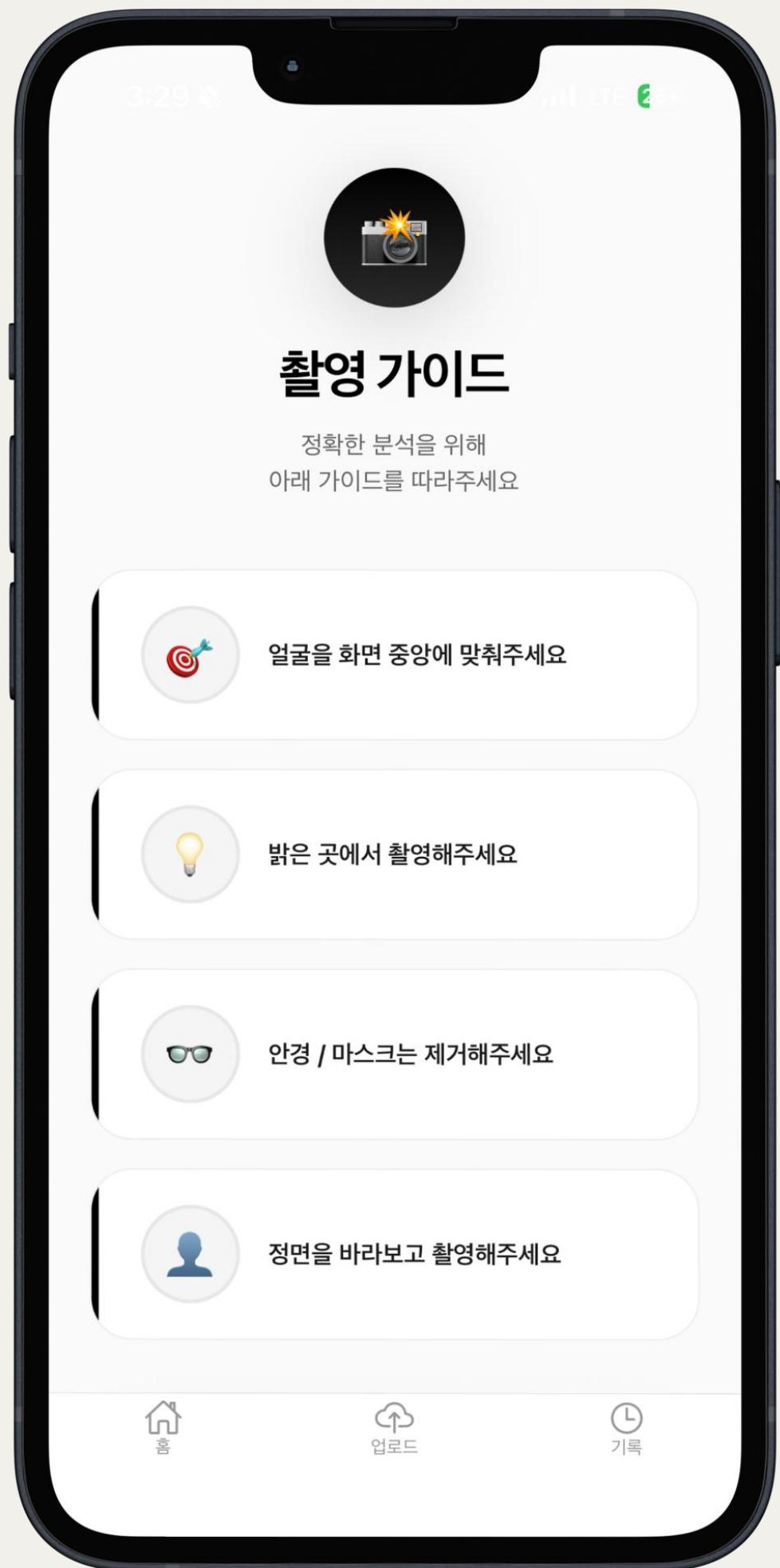
로그인 / 회원가입

- 이메일, 비밀번호 기반 로그인 처리
- JWT 액세스 토큰 발급 및 저장
- 기본 정보 입력 후 가입 요청
- 이메일 인증을 필수 단계로 설계

이메일 인증 / 프론트 처리

- 회원가입 완료 시 인증 메일 발송
- 인증 완료 전 주요 API 접근 차단
- 인증 상태에 따라 화면 및 액션 분기
- 인증 미완료 시 모달 UI로 안내





COMP TYPE 2

촬영 가이드 기능 - 기술 중심 설명

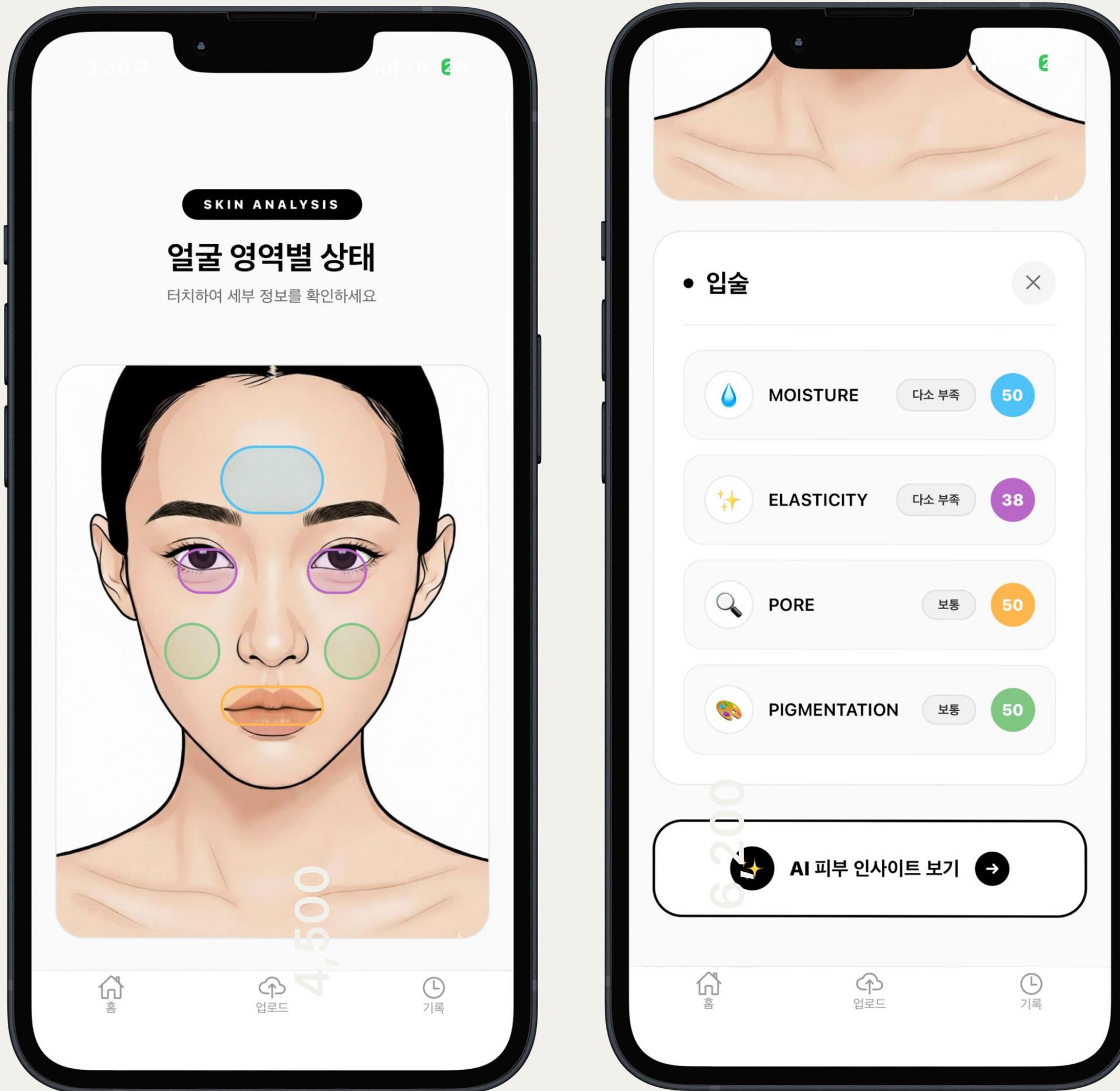
권한 및 상태 초기화 / 카메라 UI 레이어 분리

- 카메라 권한 상태를 초기 진입 시점 시점에 분기
- 권한 미획보 시 촬영 플로우 자체를 차단
- Camera Preview 와 Guide Overlay를 레이어 단위로 분리
- 오버레이는 렌더링 상태와 무관학세 유지

촬영 기준 상태 관리 / 촬영 가능 조건 제어 / 이미지 업로드 처리

- 가이드 정렬 여부를 Boolean 상태로 관리
- 조건 미충족 시 촬영 이벤트 비활성화
- 권한 / 정렬 / 준비 상태를 AND 조건으로 평가
- 모든 조건 충족 시 에만 Capture 트리거 허용
- 촬영 이미지를 Multipart 형태로 서버로 전송
- 서버에서 이미지를 AWS S3에 저장 후 접근 URL 생성
- S3에 저장된 이미지 URL을 기준으로 AI 분석 요청 및 결과 이력과 연동





COMP TYPE 3

이미지 분석 파이프 라인

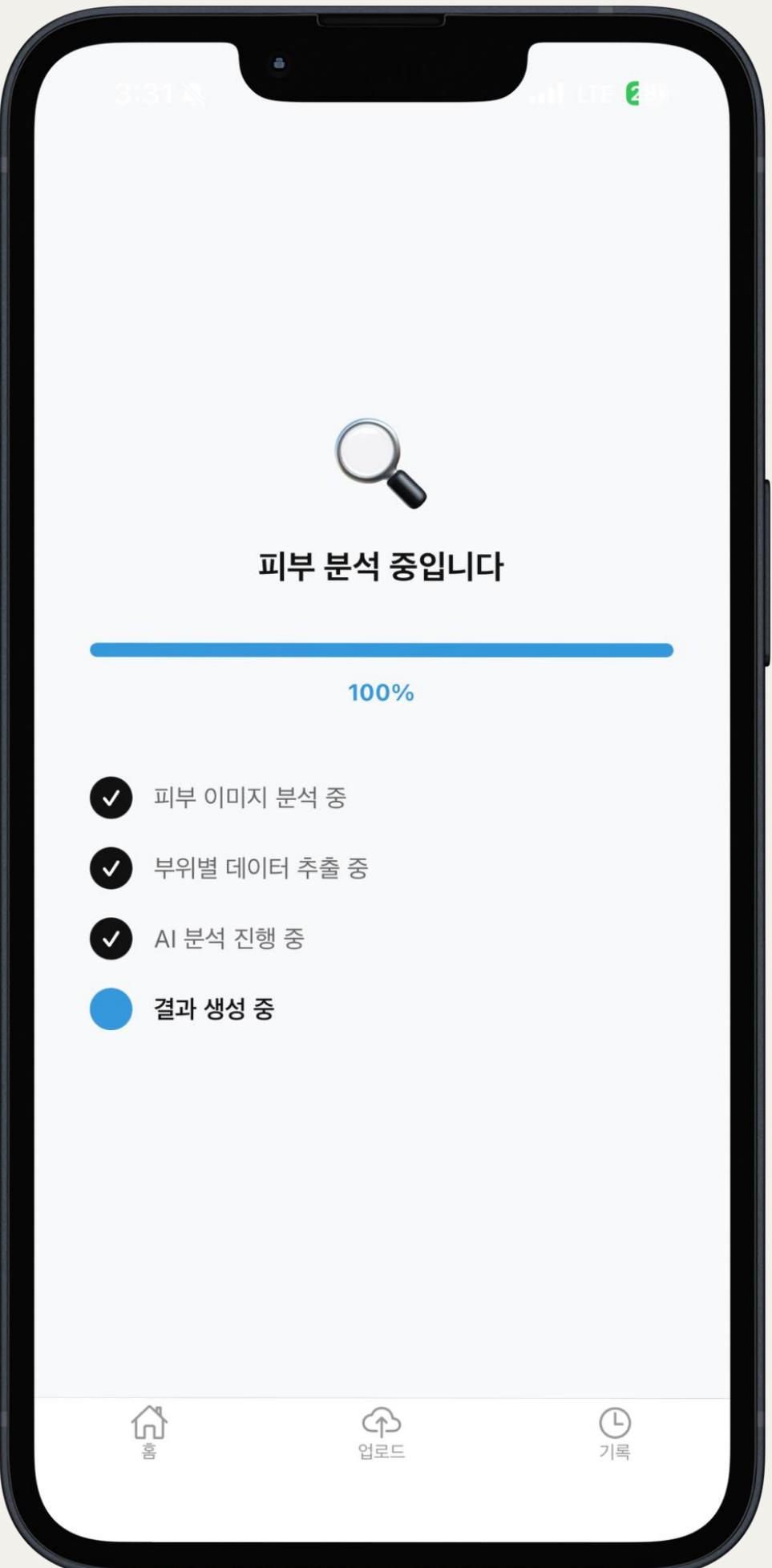
이미지 입력 기준 / 얼굴 기준 좌표 정규화

- 분석 대상 이미지는 S3에 저장된 정면 얼굴 이미지로 제한
- 촬영 단계에서 정렬을 강제하여 입력 데이터 품질을 사전 보장
- 원본 이미지 해상도와 무관하게 동작하도록 얼굴 영역을 정규화된 좌표계로 변환
- 이후 모든 부위 계산은 이 기준 좌표를 기반으로 수행

얼굴 부위 정의 방식 / ROI (Region of Interest) 단위 분리

- 얼굴의 각 부위를 얼굴 기준 비율 좌표 (hard-coded ratio)로 사전 정의
- 랜드마크 의존 없이도 안정적으로 ROI를 추출하도록 설계
☞ “모델 결과에 따라 흔들리지 않게 하려는 의도”
- 전체 이미지를 직접 분석하지 않고 부위별 ROI를 개별 이미지로 분리
- ROI 단위로 수분, 탄력, 모공, 색소 지표 계산
- 부위별 분석 결과를 0 ~ 100 범위로 정규화
- 극단값(outlier)은 상 · 하한 클amping 처리





COMP TYPE 4

인사이트 로딩 화면 – 설계 이유

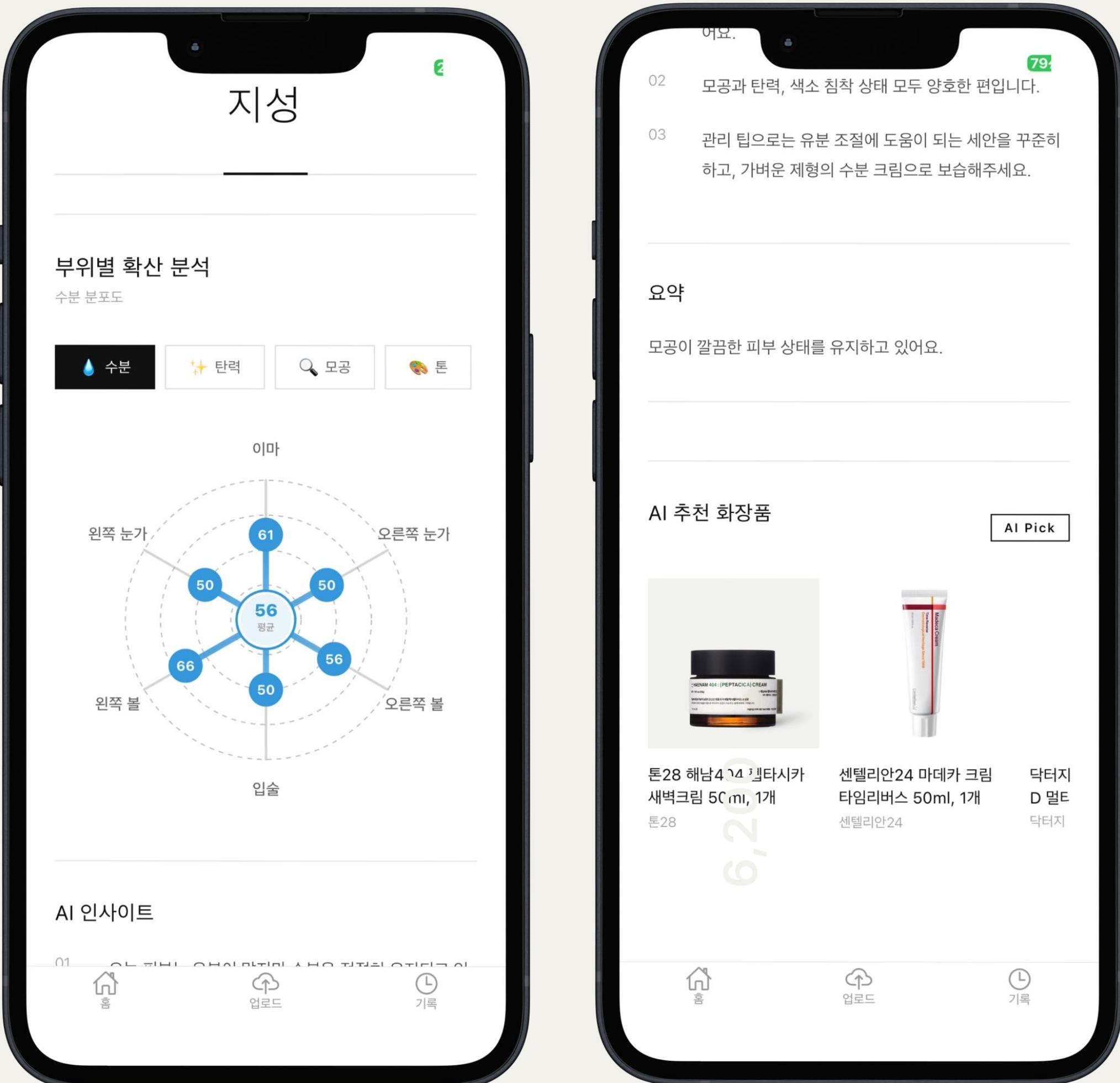
비동기 처리 분리 / 분석 지연에 대한 UX 보안

- 이미지 업로드, AI 분석, 결과 가공을 비동기 파이프라인으로 분리
- 분석 완료 시점을 사용자에게 명확히 전달하기 위한 중간 상태 필요
- AI 분석은 즉시 응답이 불가능한 작업
- 로딩 화면을 통해 "처리 중 상태를" 명시적으로 표현

분석 요청 트리고 / 분석 상태 폴링 대기

- S3 이미지 URL 기준으로 분석 API 요청
- 요청 이후 화면 전환 없이 상태 대기
- 분석 완료 여부를 상태 기반으로 확인
- 완료 시에만 결과 화면으로 이동





COMP TYPE 5

인사이트 화면에 포함 되는 내용

전체 피부 요약 / 부위별 분석 결과

- 부위별 분석 결과를 종합한 요약 정보
- 피부 타입 및 주요 상태 지표 제공
- 피부 상태는 단일 지표가 아닌 복수 지표(수분, 탄력, 모공, 톤)로 구성
- 여러 값을 한 화면에서 동시에 비교하기 위해 방사형 그래프 구조 채택

그래프 타입 선택 기준 / 부족 영역 기반 화장품 추천

- 선형 그래프는 지표 간 관계를 표현하기에 한계
- 방사형 그래프는 다차원 벡터의 분포 형태를 한 번에 표현 가능
- “값” 보다 “형태”를 인지시키는 데 최적
- 점수가 가장 낮은 지표를 1차 추천 대상으로 선정
- 분석 로직과 추천 로직을 분리 분석 결과는 순수 데이터로 유지하고, 추천은 별도 규칙 기반 레이어에서 처리
- 부족 지표 타입과 화장품 카테고리를 매핑



WEB 화면 구성

Skin Check

당신의 피부 상태를 정확하게 분석하고, 피부 타입에 맞는 화장품을 추천합니다.

피부를 체크해요

당신의 피부 상태를 체크하고 건강한 피부로 가꾸어보세요.

지금 시작하기

80% **4.9** **78%**

피부 분석 정확도 사용자 경험 평점 점수 피부 변화 추적 구조

한 눈에 확인하는 피부 분석 결과

정밀한 분석 피부 이미지를 분석해 나의 현재 피부 상태를 확인합니다.
맞춤 솔루션 피부 진단 결과에 따라 필요한 제품을 추천해드립니다.
피부 변화 추적 피부 분석 결과를 통해 피부 변화를 체계적으로 관리할 수 있습니다.

어떻게 진행되나요?

스캔하고 체크하면 끝. 3분 안에 만나보는 피부 분석 서비스

실시간 피부 스캔 시트 풀에 빠르고 정확하게 피부 상태를 분석합니다. 세안 후 스캔해 어제 풀을 바르기 전 진행하시면 가장 정확한 결과를 얻을 수 있습니다.
심층 분석 질문 더욱 정밀한 분석을 위해 생활습관, 피부 약제의 등을 확인합니다. 이 분석 결과는 충합하여 개인화된 솔루션을 제공합니다.

맞춤 결과 제공 피부 분석 결과와 함께 주요 피부 고민, 주천 약제, 생활습관 개선 방안을 제공합니다.

문의하기

궁금한점이 있으시면 언제든 문의해주세요. 빠르게 답변드리겠습니다.
hello@skincheck.com

© 2026 Skin Check. All rights reserved.

Skin Check

당신의 피부 상태를 정확하게 분석하고, 피부 타입에 맞는 화장품을 추천합니다.

당신의 피부 상태를 체크하고 건강한 피부로 가꾸어보세요.

지금 시작하기

피부를 체크해요

당신의 피부 상태를 정확하게 분석하고, 피부 타입에 맞는 화장품을 추천합니다.

당신의 피부 상태를 체크하고 건강한 피부로 가꾸어보세요.

지금 시작하기

80% **4.9** **78%**

피부 분석 정확도 사용자 경험 평점 점수 피부 변화 추적 구조

한 눈에 확인하는 피부 분석 결과

정밀한 분석 피부 이미지를 분석해 나의 현재 피부 상태를 확인합니다.
맞춤 솔루션 피부 진단 결과에 따라 필요한 제품을 추천해드립니다.
피부 변화 추적 피부 분석 결과를 통해 피부 변화를 체계적으로 관리할 수 있습니다.

x

AI 피부분석

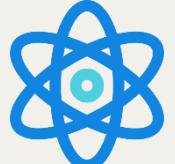
모바일 환경에서 가능합니다
휴대폰으로 QR 코드를 촬영하여 피부분석을 시작해보세요

스마트폰 카메라로 QR 코드를 스캔하시면
AI 피부분석 페이지로 이동합니다

Skills

Client (Frontend / Mobile)

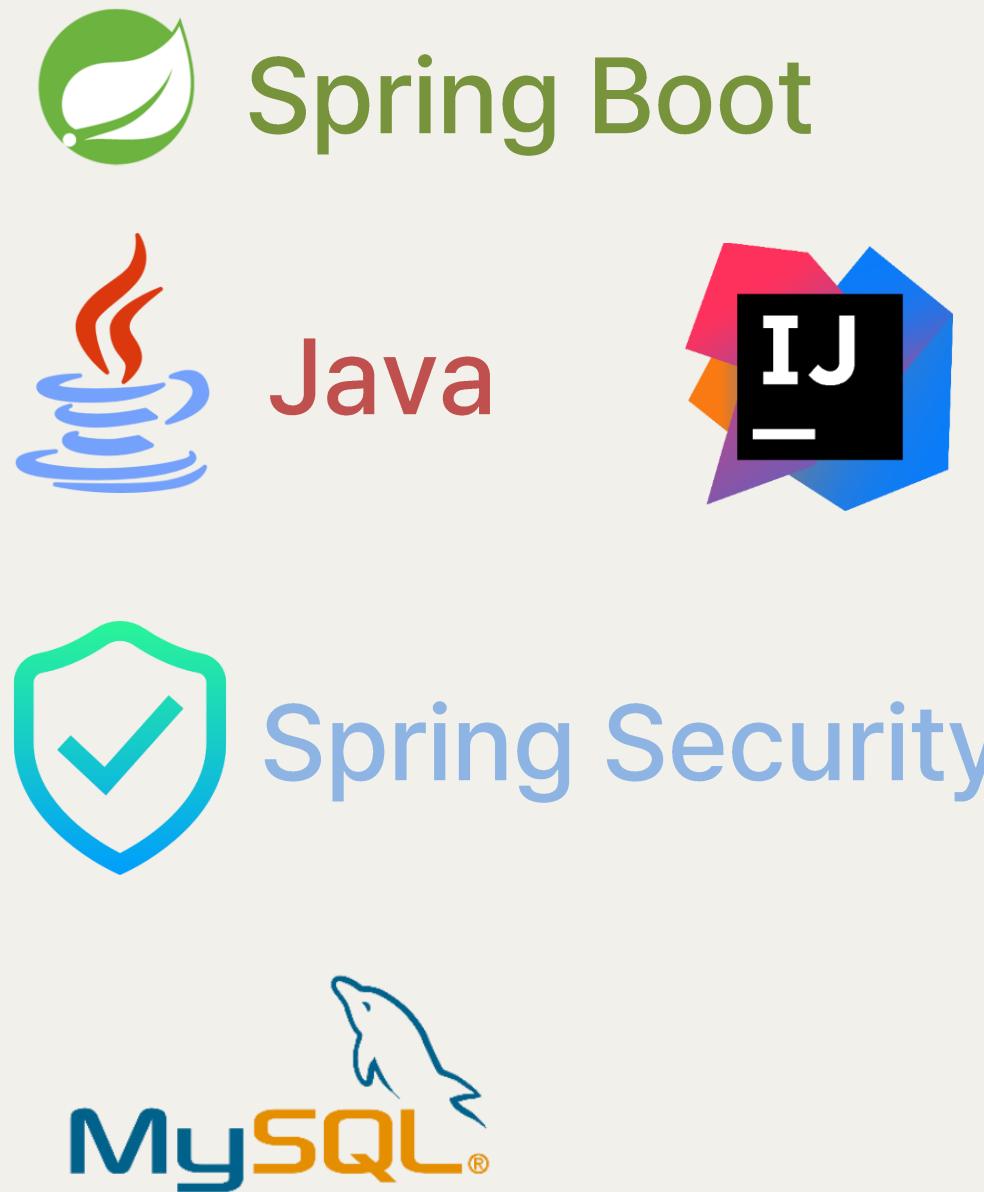


 React Native

 TS

- 웹과 모바일 클라이언트 구현
- 카메라 촬영, UI 인터랙션, 데이터 시각화 담당

Server (Backend)



- 인증 / 인가 및 비즈니스 로직 처리
- 사용자, 분석 데이터 관리
- AI 서버와 클라인트 연결 허브

AI / Analysis

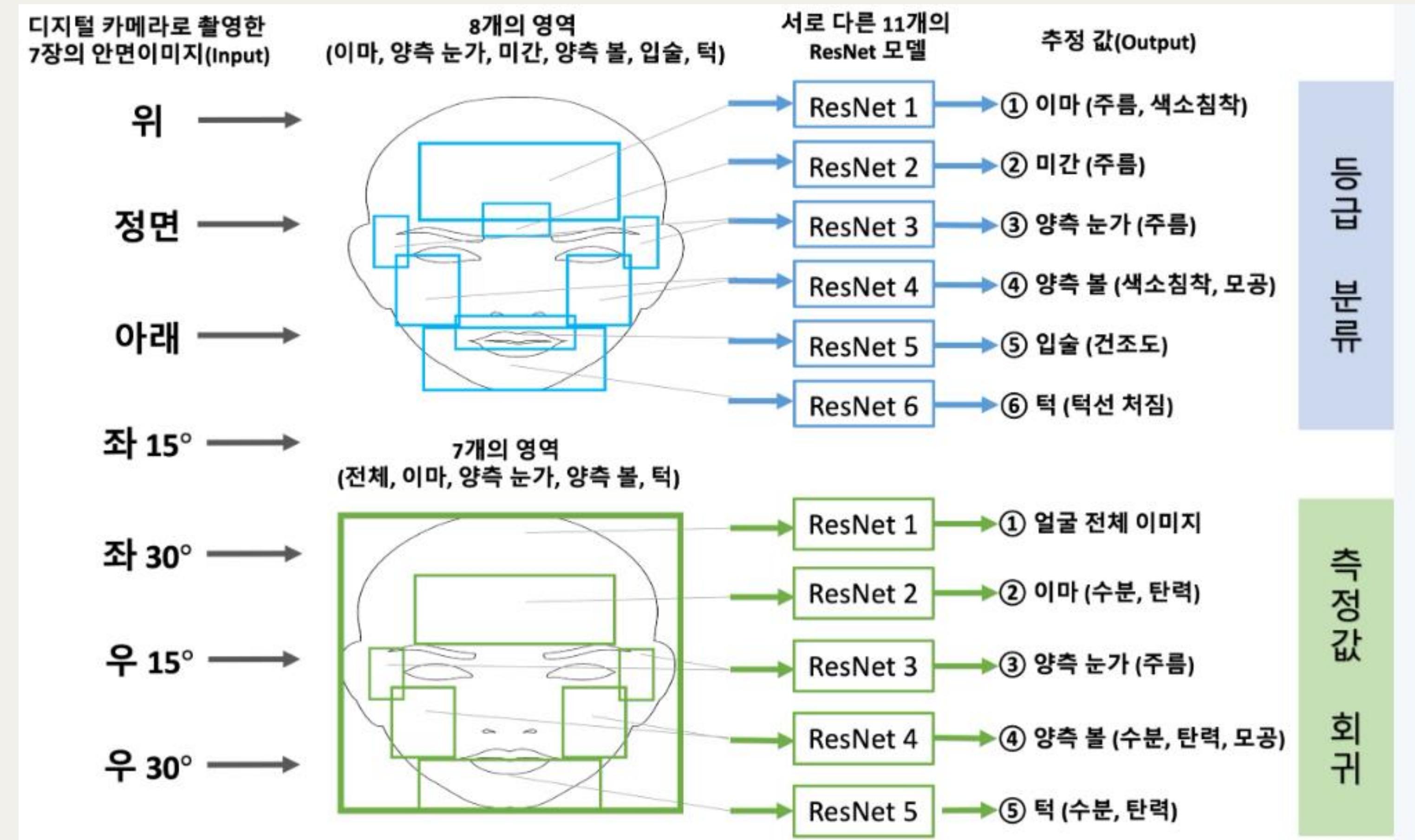


- 얼굴 이미지 분석
- 피부 상태 수치화
- 분석 결과 API 제공

개선사항

모델 개선사항

- 각 이미지 별 부위를 특정해주는 객체 탐지를 이용하여 이미지 부위별 Crop
- 해당 분류이미지를 축소된 개별 라벨링 데이터와 별개의 데이터셋으로 압축
- 서로 다른 모델 학습을 통하여 전체 회귀분석이 아닌 부위별 추론을 할 수 있게끔 구현
- 부위별 특징을 모델이 집중적으로 학습하여 개선된 추론을 기대할 수 있음



데이터셋과 모델 적합성

수행한 학습방식이 제공된 데이터셋과
맞지않아 모델 일반화에 한계가 있음



서비스

분류, 회귀 모델로 피부타입 측정에 100%
적합 하진 않음
피부 타입 측정 후 전.후 비교 서비스에 더
적합함
추후 예정 : iOS 개발, 안드로이드 개발
환경 추가 후 수정 후 배포 예정



PC환경

GPU가 없는 PC환경에서는 학습 시간이
과도하게 소요됨

iOS 개발
언어는 Swift/Objective-C 중심이며,
Windows에서 이를 직접 지원하는 공식
도구가 없음

iOS는 애플 기기·하드웨어에만 호환되는
폐쇄적 개발환경이라 Windows에서의
개발/테스트가 원활하지 않음

