# Final Report

## 컬러시즈닝(Color Seasoning)


▌지도 교수 : 김두현 교수님

▌팀원 : 201411253 홍민지

▌팀원 : 201211275 박민경


▌제출일 : 2018년 12월 7일

# 목차


1. Introduction

2. Problem

2.1 Current Solution

2.2 Objective

2.3 Functional Requirement

2.4 Nonfunctional Requirement

2.5 Target Environment

3. Requirement Analysis

3.1 Use case Model

Top-level use case diagram

Use case document for each use case

3.2 Dynamic model

Sequence diagram for each use case

4. System Design

4.1 Top level design

System flow chart

Top-level system decomposition
Describe all subsystem with explanations

4.2 Subsystems

Viewer Management
Controller Management
Database Management

## 5. Solution


5.1 Image Processing

5.2 Face Detection Algorithm

5.3 Cheek Detection

5.4 PCCS Skin Tone Structure

5.5 PCCS Tone Classification Algorithm

5.6 Tone Ranking

5.7 2-Way Recommendation


## 6. Schedule

## 7. Conclusion

## 8. Reference

# 1 Introduction

"퍼스널 컬러"는 아직 많은 남성들에겐 생소한 개념이지만 여성들의 메이크업, 패션에 적용되는 개념입니다. 퍼스널 컬러의 사전적인 의미는 '개성을 중시하는 현대 사회에서 자신의 개성을 나타내기 위해 자신에게 특유의 색상을 부여하는 것으로 사람의 머리카락이나 눈동자, 피부 톤을 종합하여 가장 어울리는 색상을 찾는 이론'입니다.

자신만의 퍼스널 컬러는 색조 화장품이나 옷, 장신구가 어울리는지 알아볼 때 사용하는데 얼굴이 하얗다고 쿨톤, 어둡다고 웜톤 이렇게 단순하게 나뉘는 것이 아니라 쿨톤은 차가운, 시원한, 도도한 느낌이며 웜톤은 따스한, 부드러운 느낌으로 구분하며 자신의 피부색이 웜톤이면 자신의 피부와 어울리는 색도 웜톤이라는 뜻이 됩니다. 한 마디로 퍼스널 컬러는 색채의 이미지화이며 색의 성질과 감정을 다루는 이론이라고 볼 수 있습니다.

이러한 퍼스널 컬러가 대두되고 있는 이유는 자신의 스타일링에서 자신의 단점은 보완해주고 장점을 빛나게 해주는 색을 찾을 수 있기 때문입니다. 얼굴형과 이목구비와 신체조건이 다 다르듯, 그 사람에게 맞는 색이 존재한다는 것입니다. 퍼스널 컬러 전문업체에 가서 진단을 받는 비용은 10만원~15만원 선으로 고비용이지만 많은 여성들이 어울리지 않은 화장품과 옷을 잘못사서 돈을 낭비하는 것보다 한번 퍼스널 컬러를 진단받고 어울리는 스타일링 제품들을 사는 것에 투자하는 것이 더 좋다고 여기는 추세입니다.

그래서 직접 가야하는 번거로움과 고비용이 드는 퍼스널 컬러 진단 법을 어플을 통해 손쉽게 진단해주고, 진단에 맞는 아이템들을 추천해 주는 모바일 프로그램을 개발하여 개인의 최상의 이미지를 연출하는데 도움을 주고 자신에게 맞는 아이템들을 위한 합리적인 소비를 도와주는 것이 목적입니다.

# 2 Problem

'퍼스널 컬러'에 많은 뷰티 업계들이 주목하고 있고, 이론이 많이 전파되고 있는데 이는 분명 '퍼스널 컬러'가 실제로 스타일링에 도움이 된다는 것을 입증해줍니다. 하지만 사람에 의한 퍼스널 컬러 진단은 다양한 특성을 반영하지 못하고 주관적인 판단에 의해 좌지우지되는 항목이 많기 때문에 정확성이 떨어집니다. 진단하는 업체나 개인마다 진단법도 다르고, 보통 얼굴 밑에 다양한 색의 진단천을 대며 색의 어울림을 판단하는데 사람마다 따뜻함과 시원함을 느끼는 온도가 다르듯 같은 색을 보더라도 진단하는 사람에 따라 그 색이 따뜻하다 시원하다고 느끼는 기준이 다르므로 개인마다 웜/쿨의 경계가 다릅니다. 또 퍼스널 컬러 이론에는 서양식과 동양식으로 나누어져 있으며 정확한 이론의 표준도 없고 회사별, 기업별 진단의 척도가 다르기 때문에 사람의 시각과 인지에 의존할 수밖에 없습니다.

따라서 정확한 진단이 존재하지 않는 퍼스널 컬러대신 "PCCS 색체계"를 알고리즘에 적용하여 자체 'SKIN TONE DETECTION'알고리즘을 개발하였으며 또 사진에 의한 진단의 한계점을 극복하기 위해 실제 피부색상과 이미지의 피부색상을 최대한 비슷하게 추출하기 위해 조명의 색과 온도, 밝기 등을 보정하기 위한 이미지처리 시스템도 어플리케이션 내에 적용시켜 사진을 통한 간단한 피부톤 진단 어플리케이션을 개발하였습니다.

## 2.1 Current solution

- 전문가에게 직접 방문하여 퍼스널 컬러를 진단받는다.
- 컬러카드를 신청하여 배송 받은 후, 컬러카드와 함께 사진을 측정하는 방식으로 퍼스널 컬러를 진단한다.
- 사용자 스스로 판단하기에 자신의 피부와 어울리는 색상을 어플리케이션 내에서 주어지는 색상 중에 선택하여 그 통계를 통해 진단한다.
- 메이크업 전의 상태로 진단할 것을 요구한다.

## 2.2 Objective

- 일단 '퍼스널 컬러'에 대한 이론은 표준화되지 않았으며, 정확히 정립되지 않은 '퍼스널 컬러'라는 기준을 없애고 PCCS 톤 체계의 톤 개념을 활용한다.
- 컬러카드를 신청하거나, 스스로 어울리는 색상을 선택하는 방식은 객관적이지 않으므로 톤 결정을 위한 객관적이고 정량화된 기준을 정립한다.
- 메이크업 전, 후 상태에서 모두 톤 측정이 가능하도록 한다.
- '퍼스널 컬러'를 어플리케이션 사용 초기에 진단을 해버리면, 퍼스널 컬러를 진단 받은 이용자는 더이상 어플리케이션을 이용하지 않을 수도 있고 어플리케이션의 사용 빈도가 줄어들 것이다.

  따라서 사용성 증대를 위하여 톤 별 화장품 추천 기능을 추가한다.

## 2.3 Functional Requirement

- 주요 기능

- 서로 다른 조명에서 5회 사진을 입력 받아, 사용자의 Tone을 진단받는다.

- 세부 기능

- 사용자에게 진단된 Skin tone에 맞는 화장품을 추천한다.
- 어플리케이션의 사용빈도와 활용도를 늘리기 위해 2개 아이템 중 사용자의 Tone에 더 적합한 Tone을 결정해준다.

## 2.4 Nonfunctional Requirement

| Category | Requirements |
| --- | --- |
| Usability | ● 간편한 사진 입력을 통해 Tone 진단이 가능하도록 한다.<br>● 많은 입력 사항은 사용자를 불편하게 할 수 있으므로 사용자는 사진만 입력 한다.<br>● 진단은 몇 번이고 다시 받을 수 있다. |
| Reliability | ● 조명의 영향을 줄이기 위해 서로 다른 조명에서의 사진을 입력 받아 각 조명에서의 결과를 종합하여 Tone을 측정한다.<br>● 제품 추천 또한 Tone 측정을 기반으로 사용자 Tone에 커스터마이징된 제품을 추천한다. |
| Performance | ● 정확한 Tone 진단이 가능하도록 한다.<br>● 여러 제품 중 사용자의 Tone에 적합한 제품을 선택해준다. |
| Packaging | ● 사용자가 처음 어플리케이션을 설치할 때 이외의 다른 설치는 없다. |

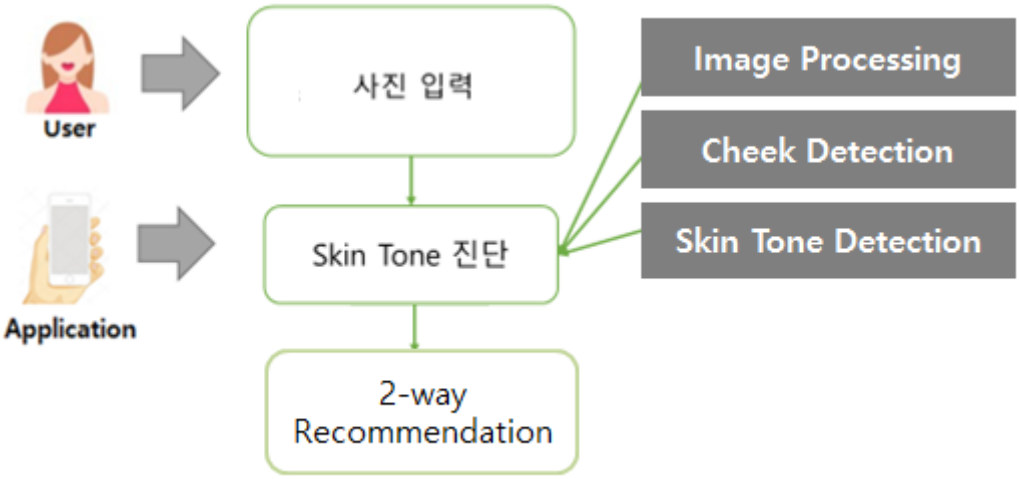
### 2.5 Target Environment

- Android 6.0.1 API 23 이상

# 3 Requirement Analysis

### 3.1 Use case model

- Top-level Use Case Diagram




- Use Case document for each use case

### ✓ Identifying Actors

| | |
|---|---|
| **Users** | 서비스를 받는 사용자들 |
| **Admin** | 어플리케이션 개발자 및 관리자 |
| **Product DB** | 화장품 데이터 |
| **Color DB** | Tone 정보를 저장하는 데이터베이스 |
| **Image Processing Algorithm** | 사진 보정을 위한 선 처리 작업을 포함한 알고리즘 |
| **합산 Algorithm** | Tone 정보를 활용하여 점수를 매기는 합산 알고리즘 |

### ✓ Identifying Scenarios

1. 초기 설정

| | |
|---|---|
| Scenario Name | 사진 및 각종 입력 사항 입력하기 |
| Participating Actor Instance | 민지 |
| Flow of Events | 1. 민지는 '컬러시즈닝' 어플을 구글스토어에서 다운로드 한다.<br>2. 서로 다른 조명에서 사진을 5회 입력한다. |


2. Tone 진단

1) 진단 결과

| Scenario Name | Tone 진단받기 |
|---|---|
| Participating Actor Instance | 민지 |
| Flow of Events | 민지는 자신의 Tone을 진단 받는다. |

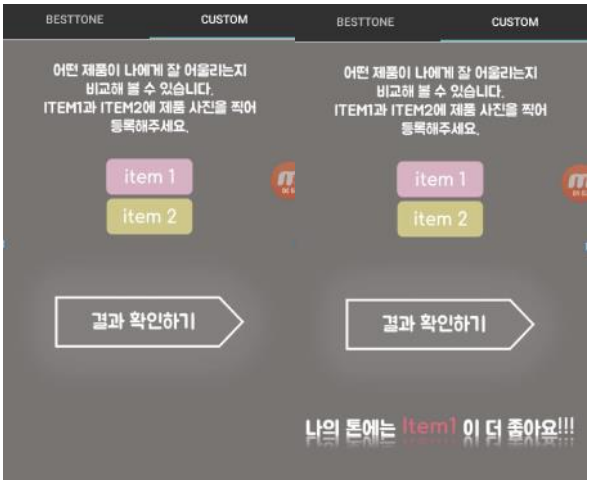

3. Tone 진단 후

1) 제품 추천

| Scenario Name | Tone별 제품 추천 받기 |
|---|---|
| Participating Actor Instance | 민지 |
| Flow of Events | 민지는 진단 받은 Personal Color에 맞는 제품을 추천 받는다. |


2) 두 제품 중 자신의 Tone과 더 적합한 제품 추천

| | |
|---|---|
| Scenario Name | 자신의 Tone과 적합한 제품 추천 받기 |
| Participating Actor Instance | 민지 |
| Flow of Events | 1. 민지는 옷, 화장품 등을 구입할 때 고민되는 색상 2가지를 사진을 찍어 입력한다.<br>2. 2가지 중 민지의 Tone에 더 적합한 제품을 추천 받는다. |




## ✓ Identifying Use cases

| | |
|---|---|
| **Use case name** | 사진 입력하기 |
| **Participating actors** | User, Color DB, Admin |
| **Flow of events** | 1. User는 퍼스널 컬러 진단을 위해 사진을 5회 입력한다.<br>- 기기의 라이브러리에 저장 된 사진 입력<br>- 바로 찍은 사진 입력 |
| **Entry condition** | User는 카메라 버튼을 눌러 사진을 찍거나,<br>사진첩 버튼을 눌러 기기의 사진을 입력한다. |

| | |
|---|---|
| **Exit Condition** | 사진을 입력 후 완료 |
| **Quality requirements** | 정확한 진단을 위해 5회 사진 입력 시 서로 다른 조명에서의 촬영을 요구한다. |

| | |
|---|---|
| **Use case name** | Tone 진단 받기 |
| **Participating actors** | User, Admin, 이미지 처리 알고리즘, 진단 알고리즘 |
| **Flow of events** | 1. Admin은 입력 받은 사진을 각각 이미지 처리 알고리즘을 적용한다.<br>2. User는 입력 받은 사진을 바탕으로 12가지의 Tone 중 1가지를 진단 받는다.<br>3. 사용자 정보에 진단받은 Tone 정보가 저장된다. |
| **Entry condition** | 사진을 Tone 진단 알고리즘에 입력한다. |
| **Exit Condition** | 알고리즘에 의한 진단 완료 |
| **Quality requirements** | User는 다른 사진으로 재진단을 받을 수 있다. |

| | |
|---|---|
| **Use case name** | 아이템 추천 받기 |
| **Participating actors** | User, Admin, Product DB |
| **Flow of events** | 진단 알고리즘은 User의 Tone을 바탕으로 데이터베이스에 저장된 뷰티 아이템을 추천해준다. |
| **Entry condition** | 진단된 결과를 추천 알고리즘에 입력한다. |
| **Exit Condition** | 알고리즘에 의한 진단 완료 |

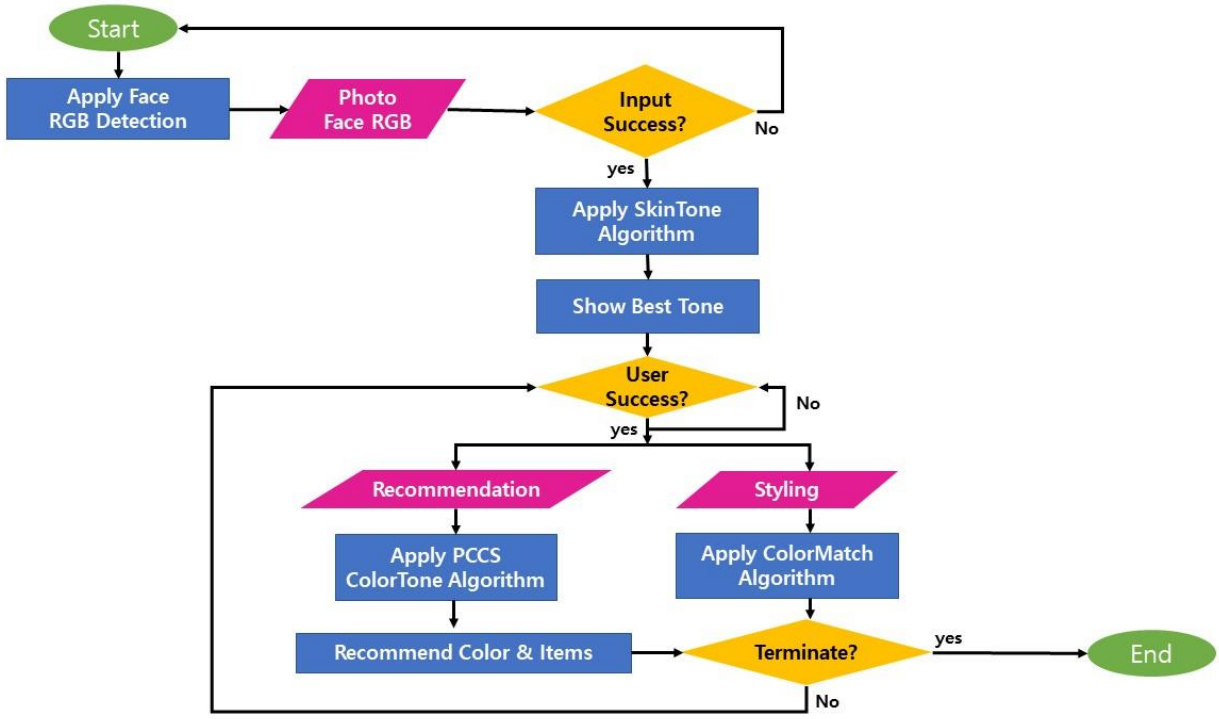
| | |
|---|---|
| **Use case name** | 2개 아이템의 색상 입력하기 |
| **Participating actors** | User, Admin, 진단 알고리즘, 합산 알고리즘 |
| **Flow of events** | 1. User는 2개 아이템의 색상을 카메라를 활용하여 입력한다.<br>2. Admin은 User가 제공한 사진을 바탕으로 진단 알고리즘을 통해 각 사진의 톤을 진단한다.<br>3. 진단된 Tone 결과를 활용하여 2개 아이템 중 1개를 추천한다. |
| **Entry condition** | User는 사진을 입력한다. |
| **Exit Condition** | 2개 중 1개 아이템을 출력한다. |
| **Quality requirements** | 입력 받은 사진을 바탕으로 Admin은 진단 알고리즘과 합산 알고리즘을 통해 |

| | |
|---|---|
| | 아이템을 결정한다. |

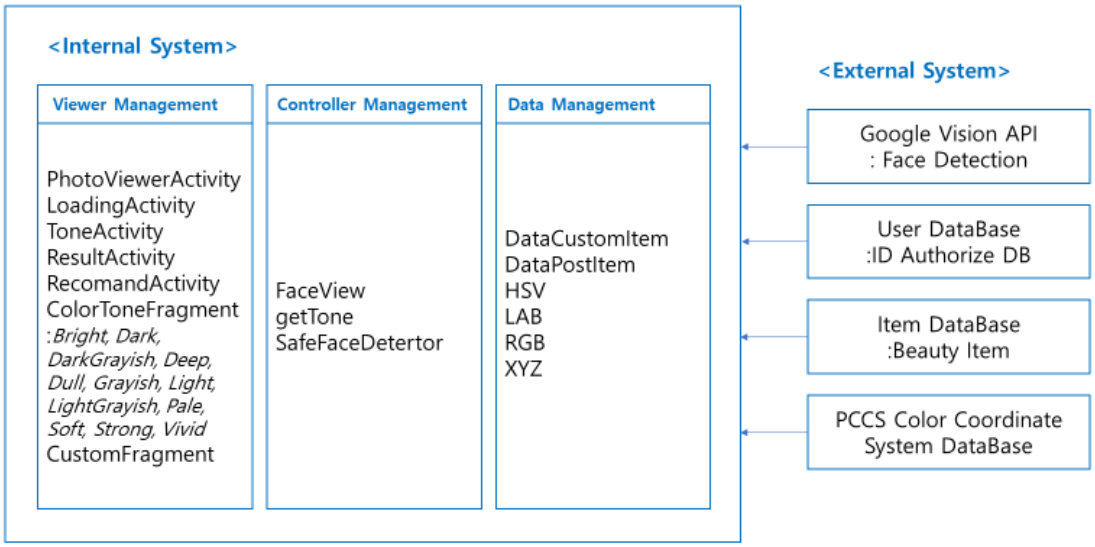
# 4 System Design

## 4.1 Top Level Design

- **System Flow Chart**




- **Top-level system decomposition**



- **Describe all subsystem with explanations**

(1) Internal System

| | |
|---|---|
| **Viewer Management** | 사용자가 Task를 수행할 때, 화면에 출력되는 외부 인터페이스들을 관리한다. |
| **Controller Management** | 이미지를 처리하거나, 스킨톤을 진단하거나, 아이템을 추천하는 등의 내부 계산을 이용하는 함수를 관리하는 클래스들의 집합을 나타낸다. |
| **Data Management** | 발생되는 데이터들과 기존의 데이터들을 관리한다. |

(2) External System

| | |
|---|---|
| **Google Vision API**<br>**: Face Detection** | 얼굴의 중요 부위 눈, 코, 입, 볼 등을 인식하는 구글 API |
| **User Database**<br>**: ID Authorize DB** | 사용자의 정보를 관리하는 데이터베이스 |
| **Item Database**<br>**: Beauty Item** | 추천할 뷰티 아이템들을 관리하는 데이터베이스 |
| **PCCS Color Coordinate System Database** | 12가지 톤의 R,G,B 값과 H, S, V 값을 관리하는 데이터베이스 |

## 4.2 Subsystem

### 1. Viewer Management

| **(1) PhotoViewerActivity** | |
|---|---|
| **설명** | 메인화면에 보여지는 인터페이스를 관리하는 액티비티이다. |
| **멤버 변수** | Button : cameraButton, galleryButton, OK, NO, tonecheck<br>View : Camera_gallery, OK_NO, howmanypicture<br>String[] tone, int time |
| **멤버 함수** | void onCreate()<br>: 로딩화면에서 메인화면 전환 인터페이스 관리, 멤버변수 초기화 및 관리,<br>void onClick(View v)<br>: 카메라 실행 혹은 갤러리 사진 요청 및 권한 허가 관리,<br>void onBackPressed(), void onResume(), void onPause()<br>void onActivityReault(Int requestCode, int resultCode, Intent data)<br>: onClick()의 요청코드에 따라 카메라 실행 혹은 갤러리에서 이미지 가져오기 실행<br>void run(Bitmap bm) |

| | |
|---|---|
| | : 가져온 이미지의 facedetect 및 RGB값 추출 요청 |

| (2) ToneActivity | |
|---|---|
| **설명** | 진단된 톤을 보여주는 인터페이스를 관리하는 액티비티이다. |
| **멤버 변수** | Button : home, reCheck<br>View : tone, good_tone, bad_tone<br>String[] tonearr, int[] tone_score, String best_tone |
| **멤버 함수** | void onCreate()<br>: 로딩화면에서 메인화면 전환 인터페이스 관리, 멤버변수 초기화 및 관리,<br>void onClick(View v), void onBackPressed()<br>void setBestTone(String[] tonearr)<br>: BestTone을 뷰에 보여주는 기능을 한다.<br>void setBadTone(String[] tonearr)<br>: BadTone을 뷰에 보여주는 기능을 한다.<br>void setScore(String[] tonearr)<br>: 5개의 리소스에서 측정된 각 톤에 점수를 매겨 평균적인 스코어를 계산하고 높은 순으로 정렬한다. |

| (3) ResultActivity | |
|---|---|
| **설명** | 결과에 의해 추천을 받을 수 있도록 전환해주는 인터페이스를 관리하는 액티비티이다. |
| **멤버 변수** | Button : brigntButton, darkButton, darkgrayishButton, deepButton, dullButton, grayishButton, lightButton, lightgrayishButton, paleButton, softButton, strongButton, vividBUtton<br>View : Camera_gallery, OK_NO, howmanypicture<br>String tone_tmp, String[] tone, |
| **멤버 함수** | void onCreate()<br>: 진단화면과 추천화면 전환 인터페이스 관리, 멤버변수 초기화 및 관리,<br>void onClickListener(View v)<br>: 버튼을 클릭했을 때 리스너가 인텐트를 통해 각 톤별 프래그먼트로 전환하여 보여주는 기능<br>void BackPressed(), void setBackgroundResource(drawble name) |

| (4) RecommandActivity | |
|---|---|
| **설명** | 2가지 방법으로 아이템을 추천받을 수 있는 인터페이스를 관리하는 액티비티이다. |
| **멤버 변수** | ViewPager viewPager, TabLayout tablayout, Fragment[] arrFragments<br>String get_tone_name |

| 멤버 함수 | void onCreate()<br>: 로딩화면에서 메인화면 전환 인터페이스 관리, 멤버변수 초기화 및 관리,<br>void getToneName(String get_tone_name)<br>: 프래그먼트 0번지에 가져온 톤의 프래그먼트를 설정한다.<br>Class MainPagerAdapter, CharSequence getPageTitne(int position), Fragment getItem(int position), int getCount() |
|---|---|

| (5) ColorToneFragment | |
|---|---|
| 설명 | 12가지 톤별 추천 아이템을 보여주는 프래그먼트이다. |
| 멤버 변수 | arrayList<DataPostItem> arrayList<br>View(Text, Recycle, Image) : baseBiew, recyclerView, vt_usertone, tv_postitemcategory, tv_iteminfo, iv_postimg |
| 멤버 함수 | View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle saveInstancdState)<br>Class PostViewAdapter, PostViewHolder(View itemView), onCreateViewHolder(ViewGroup parent, int view Type), void onBindViewHolder(PostViewHolder holder, int position), int getItemCount(), Class PostViewHolder<br>: 모든 아이템을 담을 배열 생성 및 초기화, 아이템 url주소, 이름, 설명 등을 설정하고 아이템을 인터페이스에 설정 및 관리 기능을 한다. |

| (6) CustomFragment | |
|---|---|
| 설명 | 2-wayrecommandation 중 사용자가 직접 등록한 아이템을 비교 분석하여 사용자의 톤에 더 맞는 아이템을 추천해주는 인터페이스를 관리하는 프래그먼트이다. |
| 멤버 변수 | String tone, String[] tonearr<br>RGB resultRGB_item1, resultRGB_item2<br>String[] tone_item1, tone_item2<br>int score_item1, score_item2, better_item<br>View baseView, recyclerView |
| 멤버 함수 | View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle saveInstancdState)<br>Class PostViewAdapter, Class PostViewHolder<br>PostViewHolder onCreateVIewHolder(ViewGroup parent, int viewType)<br>void onBindViewHolder(final PostViewHolder holder, int position)<br>: 두 가지 아이템의 사진을 입력 받고 진단된 스킨톤과 더 가까운 아이템을 보여주는 인터페이스를 관리하는 함수<br>void onActivitiResult(int requestCode, int resultCode, Intent data)<br>: Item의 사진을 요청코드에 따라 카메라 혹은 갤러리에서 가져오고 Bitmap변수에 저장하 |

| | |
|---|---|
| | 고 관리하는 함수<br>void checTone_item1(RGB resultRGB_item1)<br>: Item1의 RGB값으로 톤을 측정하는 함수<br>RGB getRGB(Bitmap bm)<br>: 받아온 이미지의 Bitmap에서 색을 추출하는 함수 |

## 2. Controller Management

| (1) FaceView | |
|---|---|
| **설명** | 가져온 비트맵의 이미지 처리를 하고 얼굴 중요부위를 랜드마크하고, 값을 추출하는 등의 처리하는 함수들로 이루어진 클래스 |
| **멤버 변수** | Bitmap mBitmap, adjBitmap<br>SparseArray<Face> mFaces<br>String tone[], Context mContext, int mTime |
| **멤버 함수** | Bitmap ImgProcess(Bitmap mBitmap)<br>: 조명의 색과 온도 밝기 등을 자연계 색상과 비슷하게 맞추기 위한 이미지처리를 하는 함수<br>String[] getTone()<br>: 얻어진 스킨톤을 반환하는 함수<br>void setContent(Bitmap bitmap, SparseArray<Face> faces, int time)<br>: 비트맵의 백그라운드를 설정하고 얼굴인식 변수를 설정<br>void onDraw(Canvas canvas)<br>: 비트맵에서 인식된 얼굴 중요 부위를 표시하는 함수<br>double drawBitmap(Canvas canvas)<br>: 비트맵의 사이즈를 스케일하고, 얼굴 부위를 반환하고 값을 조정하는 함수<br>voidFacdAnnotations(Canvas canvas, double scale)<br>: 랜드마크 된 부위의 좌표를 관리하며 볼 부분의 색상 값을 추출하고 반환하는 함수 |

| (2) getTone | |
|---|---|
| **설명** | 추출된 볼 부위의 RGB값을 XYZ,LAB,HSV로 변환하여 처리하는 클래스 |
| **멤버 변수** | RGB rgb, String[] tonetmp<br>double X,Y,Z, L,a,b, H,S,V |
| **멤버 함수** | getTone(RGB rgb), String[] getToneRanking(), void ToneDetection()<br>void checkTone(HSV hsv)<br>: FaceView클래스에서 추출된 색상 값들로 H,S,V의 값을 계산하여 각 톤의 S,V값과 대조하 |

| | |
|---|---|
| | 여 거리를 구해 톤을 추출하는 함수<br>HSV RGBtoHSV(RGB rgb)<br>: FaceView클래스에서 추출된 RGB값을 HSV로 변환하는 함수<br>XYZ RGBtoXYZ(RGB rgb)<br>: RGB값을 HSV로 변환하여 채도와 명도의 중간 값을 구하기 위해 RGB를 XYZ로 변환하는 함수<br>LAB XYZtoLAB(XYZ xyz)<br>: RGB값을 HSV로 변환하여 채도와 명도의 중간 값을 구하기 위해 XYZ를 LAB로 변환하는 함수 |

| **(3) SafeFaceDetertor** | |
|---|---|
| **설명** | 얼굴검출 함수에서 발생할 수 있는 에러들을 해결하는 클래스 |
| **멤버 변수** | Detector<Face> mDelegate |
| **멤버 함수** | SafeFaceDetertor(detector<Face> delegate)<br>SparseArray<Face> detect(Frame frame)<br>: FaceDetector 인스턴스와 래핑하여 사용하며 얼굴 검출기에서 kMinDemension<147인 이미지와 kDemesionLower < 640인 이미지가얼굴 검출함수와 충돌을 일으키는 것을 결정하고 충돌을 일으키는 이미지의 경우 padding을 추가하는 함수<br>Boolean isOperational(), Boolean setFocus(int id)<br>Frame padFrameRight(Frame originalFrame, int newWidth)<br>: 원본 프레임을 기반으로 오른쪽에 프레임을 추가하여 새 프레임을 만든 함수<br>Frame padFrameBottom(Frame originalFrame, int newHeight)<br>: 원본 프레임을 기반으로 아래쪽에 프레임을 추가하여 새 프레임을 만든 함수 |

## 3. Database Management

| **(1) DataCustomItem** | |
|---|---|
| **설명** | CustomFragment의 데이터들을 관리하는 클래스 |
| **멤버 변수** | Int customImg, item1, item2, item_result |
| **멤버 함수** | DataCustomItem(int customImg, int item1, int item2, int item_result),<br>Int getCustomImg(), int getItem1(), int getItem2(), getItem_result() |

| **(2) DataPostItem** | |
|---|---|
| **설명** | ColorToneFragement에서 포스팅할 아이템 데이터들을 관리하는 클래스 |
| **멤버 변수** | Int postImg, String postItemInfo, userTone, postItemCategory |
| **멤버 함수** | DataPostIem(int postImg, String postItemInfo, String userTone, String postItemCategory) |

| | |
|---|---|
| | Int getPostImg(), String getPostItemInfo(), String getUserTone(), String getItemCategory() |

| (3) HSV, LAB, RGB, XYZ | |
|---|---|
| **설명** | HSV, LAB, RGB, XYZ 값들을 받아오고 반환하는 클래스들 |
| **멤버 변수** | Doble H,S,V, L,a,b , R,G,B, X,Y,Z |
| **멤버 함수** | double getH(), getS(), getV()<br>double getL(), getA(), getB()<br>double getR(), getG(), getB()<br>double getX(), getY(), getZ() |

# 5 Solution

- **Important Obstacles**
  - 사진은 조명의 색과 온도와 밝기에 영향을 받는데 실제 색과 비슷한 피부색상을 추출하기 위해 이미지처리가 필요하다.
  - 추출된 피부의 색상을 특정 톤으로 매칭시키기 위한 정량적인 이론이 필요하다.
  - 얼굴에서 가장 빛을 잘 받는 부위인 이마 중앙, 턱 중앙, 볼 중앙 중 어느 부위를 선정할지 어떤 방법으로 구현할지 결정하는 것이 필요하다.
  - 피부톤과 아이템의 색상 값을 구조화하는 것이 필요하다.
  - 특정 아이템과 톤을 매칭시키고 추천해주는 알고리즘을 구현해야 한다.

- **Our Solution**
- 이미지 보정을 위한 Adaptive Histogram Equalization 알고리즘 구현
- 얼굴 피부색 부분 검출을 위한 Face Detection 알고리즘 구현
- 검출된 피부색의 스킨톤을 정하기 위한 Skin Tone Detection Algorthm 설계
- 기준 부위를 볼로 측정
- PCCS기법을 이용한 피부 톤 자료구조 구현
- 화장품을 톤 별로 분류하기 위한 PCCS 톤 분류법

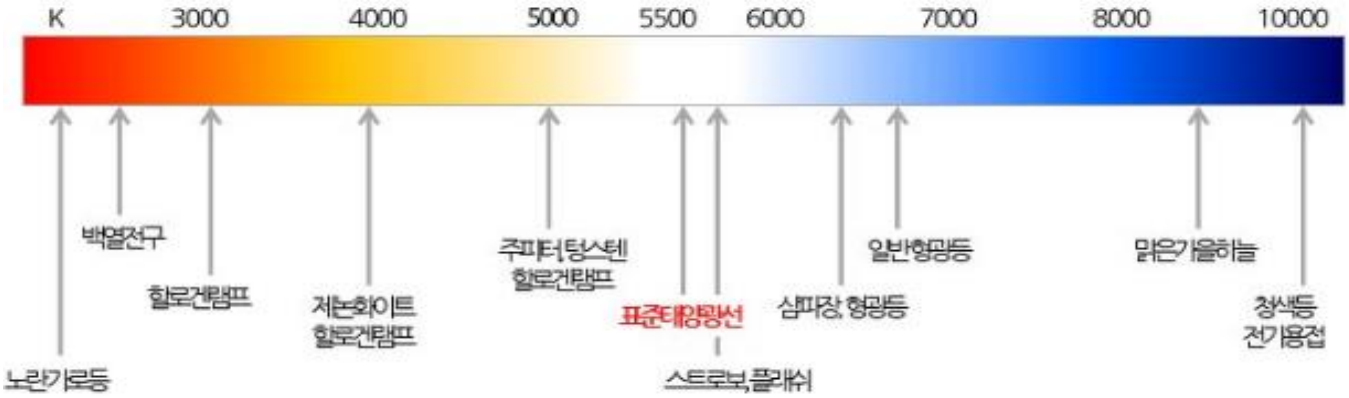
## 5.1 Image Processing

이미지를 인식할 때 영상 촬영 환경 및 조건에 따라서 매우 다양한 색상의 이미지가 나올 수 있다. 예를 들어 맑은 날, 흐린 날, 비 오는 날, 그늘에 가렸을 때, 그늘이 없을 때, 조명, 역광, 한낮, 해질 무렵에 따라 영상의 색이 모두 다르고 동일한 카메라라고 할 지라도 화이트 밸런스로 인해 색이 변할 수도 있다. 이렇게 복잡하고 다양한 색상 변화 요인에 대처하기 위한 두 가지 방법 중, 하나는 색은 믿을 수 없으니 아예 색을 사용하지

않고 밝기 정보 값 만을 사용하는 방법과 두 번째는 밝기 정보는 무시하고 순수한 색상 정보만을 이용하는 방법이 있다. 밝은 곳과 어두운 곳에서 변하는 색상차는 밝기 값 때문이지 색 자체에는 변함이 없다는 것이다.

- 화이트 밸런스 조정 : AWB(Auto White Balance)

얼굴을 형광등 아래에 찍었을 경우와 백열 전구 아래에서 찍었을 때 색감이 다르게 찍힌다. 화이트 밸런스는 색의 온도가 낮은 붉은색 광원을 푸른색을 섞어 보정하고 색 온도가 높은 푸른색 광원은 붉은색을 섞어 보정하고 표준 태양광선의 온도(5500~5600K)로 맞춰 색감의 차이를 조절해주고 실제 피부색을 추출하도록 해준다.




화이트 밸런스를 처리하는데 기본적으로 이용되는 필터는 베이어 필터(Bayer Filter)이다. 베이어 필터는 한 개의 R, B와 두개의 G로 이루어진 패턴이다. G는 사람의 눈이 녹색 값에서 가장 높은 감도를 보이고 자연색과 가까운 색으로 2개의 공간을 가지는데 여기서 G는 일반적인 녹색이 아닌 녹색과 더불어 휘도(Y, Luminance)의 의미를 가진다. RGB->YCbCr 변환 공식 중 가장 많이 사용되는 공식은

```
Y  = ( 0.2290 * R) + ( 0.5870 * G) + ( 0.1140 * B) + 128
Cb = (-0.1687 * R) + (-0.3313 * G) + ( 0.5000 * B) + 128
Cr = ( 0.5000 * R) + (-0.4187 * G) + (-0.0813 * B) + 128
```

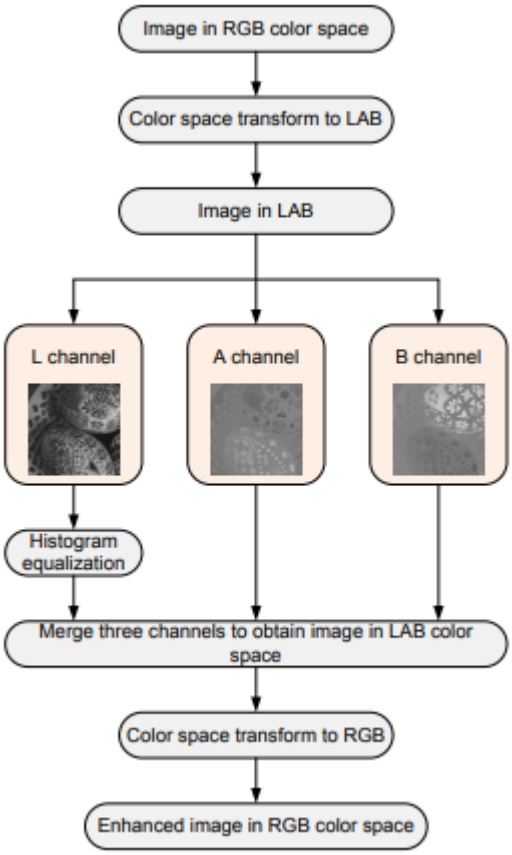
이며 Cb, Cr에 초점을 맞춘 식은 다음과 같다.

```
Cb = (0.5 * (B - G)) - (0.1687 * (R - G)) + 128
Cr = (0.5 * (R - G)) - (0.0813 * (B - G)) + 128
```

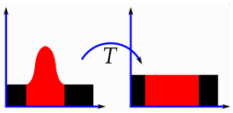
여기서 B, G 성분에서 휘도인 G 성분을 뺀 값이 절반 이상의 영역(0.5*)을 차지하고 있다. 따라서 Cr은 R-Y, Cb는 B-Y라는 형태로 쓰이고 영상의 색차 신호라 한다. AWB를 위해 사용되는 색 공간은 휘도 신호와 색차 신호를 가지는 Y, R-Y, B-Y의 정보를 가지고 표현된다.

신호와 색차 신호를 가지는 Y, R-Y, B-Y의 정보를 가지고 표현된다.

- Adaptive Histogram Equalization




**Histogram Equalization**

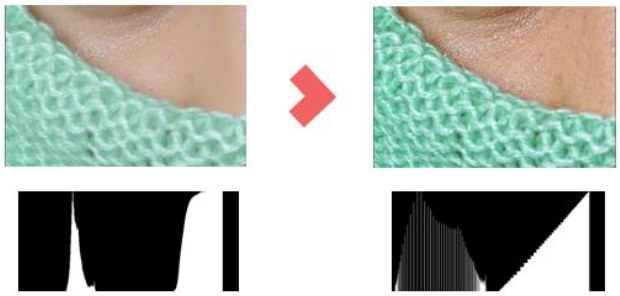



- LAB Color Space의 L 처리
- 특정 영역에 집중되어 있는 분포를 골고루 분포하도록 함
- 방법

$$\left(\frac{n_0 + n_1 + \ldots + n_i}{n}\right)(L-1)$$ 를 가장 근접한 정수로 반올림

0, 1, 2, ... , L-1 : L개의 gray level
$n_i$ : gray 값 i의 빈도 수
n : 전체 화소 수 ($n_0 + n_1 + \ldots + n_i$)

**Adaptive Histogram Equalization 결과**



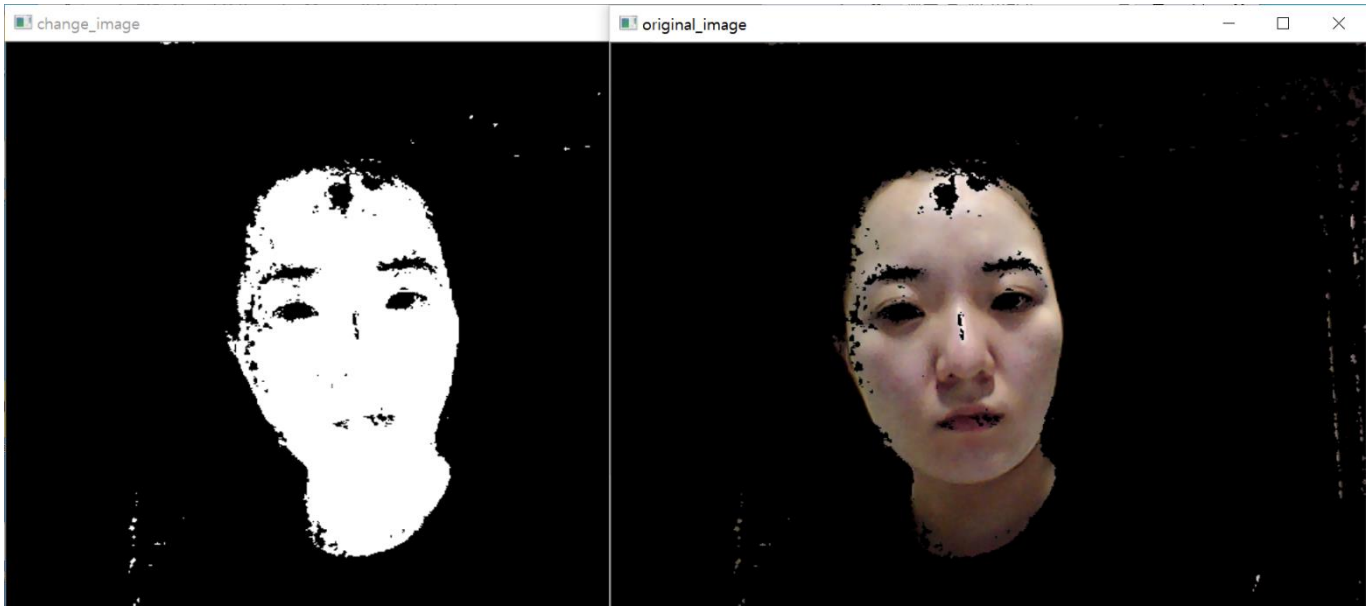
중앙에 몰려 있는 Histogram

Histogram Equalization이 적용됨

RGB의 값을 LAB로 변환 한 후 L(Lightness) 값에 Histogram Equalization을 적용 후 다시 RGB로 변환한다. 그 결과 입력 영상의 픽셀들이 중앙에 몰려 있는 히스토그램을 Histogram Equalization 적용 후 넓게 분포한 모습이다.

## 5.2 Face Detection Algorithm

하지만 우리는 어플리케이션을 이용해 피부색을 촬영할 때 대부분은 실내가 주요 촬영 장소라고 생각하여 조명이 색상변화에 큰 요인을 미칠 것이라 생각하므로 색상정보를 꼭 반영해야 한다. 특히 영상에서 피부색을 검출할 때, 사람에 따라 피부색이 어두운 사람, 밝은 사람이 있지만 모든 인간은 공통적으로 붉은 색의 피가 흐르기 때문에 피부 밝기에 관계없이 붉은색 계열의 색을 포함한다. 피부색 검출을 위해 밝기 정보를 제거하고 순수한 색상정보만을 이용하는 것이 훨씬 효과적이므로 RGB 값을 이용한다. HSV와 YCbCr 모델은 색상과 분리되어 있어 쉽게 색 정보만 활용할 수 있다. RGB모델도 normalized rg를 이용하면 밝기 정보를 제거할 수 있다. (r = R/(R+G+B), g = G/(R+G+B)) 순수 색 정보만 이용할 경우 밝기 값이 어두운 경우에는 노이즈로 생각해도 될 작은 차이가 엄청난 색 차이로 나타날 수 있다. 따라서 일정한 밝기 값이 측정되지 않으면 재 촬영을 하도록 설계했다.




- 얼굴(피부색) 영역만 검출 소스코드 적용

아시아계 황인종 피부색 검출 값
YCbCr 컬러 공간 : (Cb : 77~127 / Cr : 133~173)
RGB 컬러 공간 : R>95 && G>30 && B>20 && max{R,G,B}-min{R,G,B}>15 && |R-G|>15 && R>G && R>B

## 5.3 Cheek Detection

- 볼 부위를 기준 부위로 지정

-얼굴의 여러 부위 중 기준이 되는 부위를 볼 부위로 지정하기 위한 근거 정립이 필요
-메이크업을 하지 않은 피부 상태 측정을 요구하려 하였으나, 활용도가 떨어질 수 있음
-따라서 메이크업을 하여도 얼굴의 어느 부위의 색이 주로 보정되지 않는지 파악
-비교 대상은 기존 피부 색 측정의 보기였던 얼굴의 제일 튀어나온 부분, 즉 이마 중앙, 볼 중앙, 턱 중앙으로 설정
-20명의 지인 사진 메이크업 전 후 사진 활용
-이미지의 RGB를 측정하였지만, 사람의 색 표현 정량화에 적합한 CIE L*a*b Color space로 변환하여 메이크업 전 후를 비교하는 것이 적절하다고 생각하여, RGB를 CIE L*a*b Color space로 변환함
- RGB → XYZ → CIE LAB

```
var_R = ( sR / 255 )
var_G = ( sG / 255 )
var_B = ( sB / 255 )

if ( var_R > 0.04045 ) var_R = ( ( var_R + 0.055 ) / 1.055 ) ^ 2.4
else                   var_R = var_R / 12.92
if ( var_G > 0.04045 ) var_G = ( ( var_G + 0.055 ) / 1.055 ) ^ 2.4
else                   var_G = var_G / 12.92
if ( var_B > 0.04045 ) var_B = ( ( var_B + 0.055 ) / 1.055 ) ^ 2.4
else                   var_B = var_B / 12.92

var_R = var_R * 100
var_G = var_G * 100
var_B = var_B * 100

X = var_R * 0.4124 + var_G * 0.3576 + var_B * 0.1805
Y = var_R * 0.2126 + var_G * 0.7152 + var_B * 0.0722
Z = var_R * 0.0193 + var_G * 0.1192 + var_B * 0.9505
```

RGB → XYZ

```
var_X = X / Reference-X
var_Y = Y / Reference-Y
var_Z = Z / Reference-Z

if ( var_X > 0.008856 ) var_X = var_X ^ ( 1/3 )
else                    var_X = ( 7.787 * var_X ) + ( 16 / 116 )
if ( var_Y > 0.008856 ) var_Y = var_Y ^ ( 1/3 )
else                    var_Y = ( 7.787 * var_Y ) + ( 16 / 116 )
if ( var_Z > 0.008856 ) var_Z = var_Z ^ ( 1/3 )
else                    var_Z = ( 7.787 * var_Z ) + ( 16 / 116 )

CIE-L* = ( 116 * var_Y ) - 16
CIE-a* = 500 * ( var_X - var_Y )
CIE-b* = 200 * ( var_Y - var_Z )
```

XYZ → CIE L*a*b

- 결과

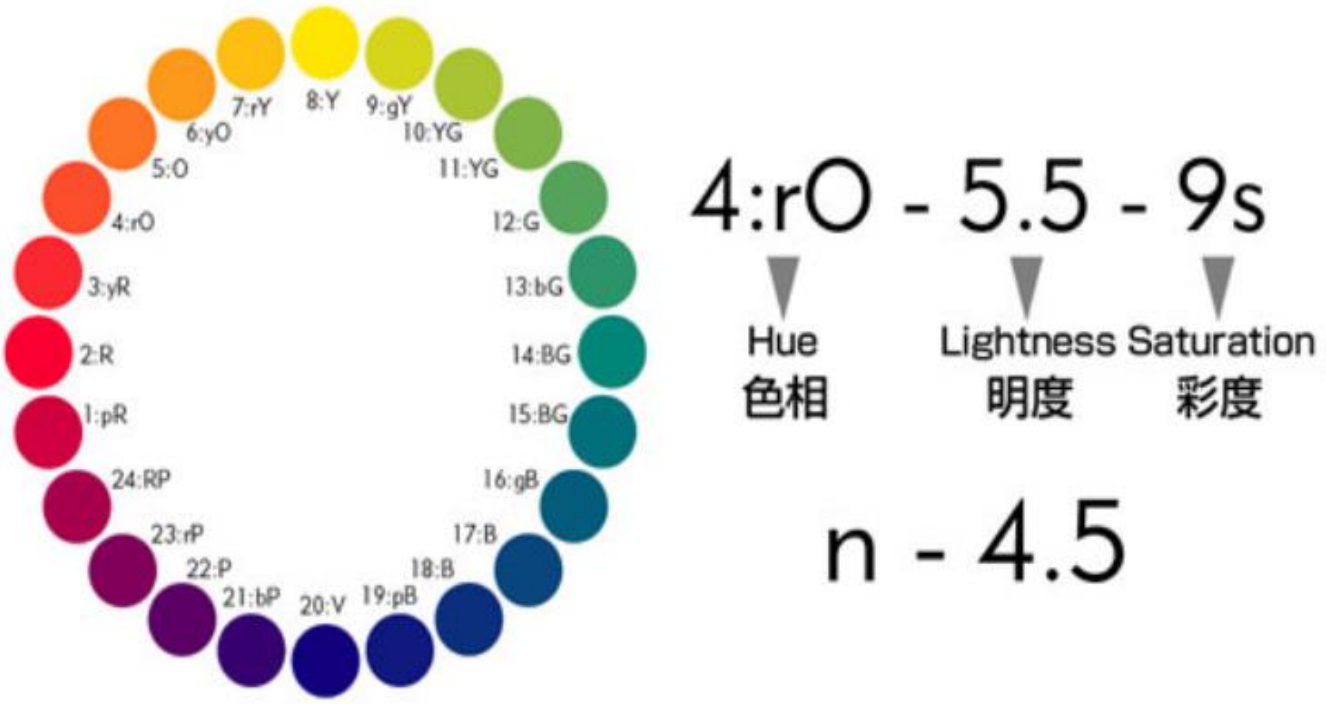
| | 측정 부위 | 메이크업 전 | 메이크업 후 |
|---|---|---|---|
| L* | 이마 | 62.04 | 62.50 |
| | 볼 | 64.20 | 64.59 |
| | 턱 | 63.61 | 64.17 |
| a* | 이마 | 09.48 | 09.47 |
| | 볼 | 09.10 | 09.20 |
| | 턱 | 10.05 | 10.00 |
| b* | 이마 | 18.66 | 19.42 |
| | 볼 | 17.77 | 17.68 |
| | 턱 | 18.36 | 17.81 |

-L*(명도) 부분은 모든 부위가 모두 차이가 크다.
a*(적색도) 부분은 모든 부위가 모두 차이가 없다.
b*(황색도) 부분을 보면, 메이크업 전 후에 볼 부분의 차이가 제일 적음을 알 수 있다
따라서, 동양인은 피부 메이크업을 할 때, 명도와 황색도 부위를 보정하는 방향으로 메이크업을 함을 알 수 있다.
또한 볼 부위를 얼굴의 여러 부위 중 기준이 되는 부위로 지정하였다.

## 5.4 PCCS Skin Tone Structure

- PCCS(Practical Color Coordinate System) 색 체계

'TONE(색조)'의 개념을 정립하고 색조를 색공간으로 설정하여 색상과 톤의 두 계열로 기본적 색채계열을 나타내는 방식이다.

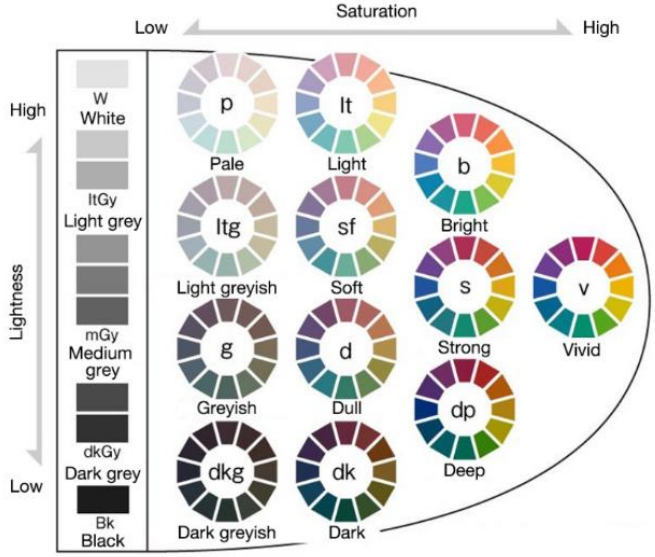



톤은 명도와 채도의 복합 개념으로 색의 상태 차이를 말한다. 색의 상태 차이는 같은 색상이라고 하여도 명, 암, 강, 약, 농, 염, 얕음, 깊음의 정도 차이를 나타낸다.

현재 퍼스널 컬러를 정량화 하는 방식은 [L*a*b* Color Space]를 이용한다. 특히 황색도(b*)에 주목하여 피부톤을 분류하지만, 우리는 피부톤과 색상 매칭을 위해 PCCS색체계를 이용하여 피부색을 분류한다.

- PCCS Tone

PCCS 톤은 pale(아주연한), light(연한), bright(밝은), vivid(해맑은), strong(기본색), soft(부드러운), dull(칙칙한), deep(짙은), dark(어두운), light grayish(연한 회), grayish(회), dark grayish(어두운 회)의 12 종류로 나누어진다. 따라서 각 색상마다 12 가지 톤으로 나누어, 같은 톤으로 색을 묶으면 명도의 차이는 있지만 뚜렷하게 느껴지는 공통의 그룹이 가능하다.




<PCCS 색 체계>

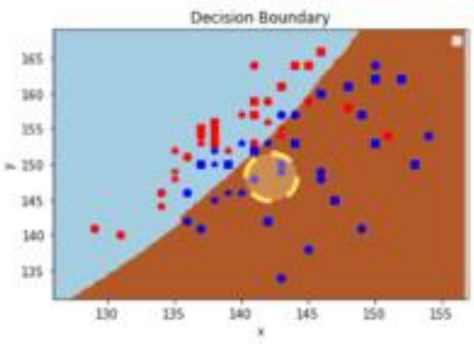
PCCS에서 톤을 12가지로 분류하는 방식은 명도(Value), 채도(Saturation)을 이용한다. PCCS의 각 영역의 중간 값을 추출한 후 추출한 피부색의 S, V를 이용하여 거리 계산을 통해 어떤 영역에 속하는지 확인하였다.
(이때, S(채도)값은 255를 곱하여 스케일링)

- 피부톤을 PCCS 색 체계로 톤 분류

```
// SkinTone RGB 를 HSV 로 변환
im = cv2.imread(filename+"."+img_type)
hsv_colors = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
```

<OpenCV를 이용한 피부영역 추출>

<추출 된 볼 피부색>




RGB to HSV

S = 1-3/(R+G+B)[min(R,G,B)]

V = 1/3(R+G+B)

RGB : (248.0, 195.5, 168.5)

SV : (0.17, 203.66)

SV : (0.17, 203.66)


PCCS표기법에 따라 톤 표기


$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

PCCS 톤의 중간 값과 측정한 skin tone 의 S, V 값의 거리를 활용하였다.
거리가 제일 가까운 톤 분류를 사용자의 best tone 으로 진단하고, 거리가 제일 먼 톤 분류를 사용자의 worst tone 으로 진단한다. best tone 외에 거리가 가까운 쪽에 속하는 톤 분류를 활용하여 good tone 으로 진단하여 활용하였으며, 각종 화장품 아이템의 색상톤을 판별하기 위해서도 위 방법을 적용하였으며, 사용자의 진단된 스킨톤에 더 어울리는 아이템을 추천해 줄 때도 이와 같은 방법을 적용하였다.

## 5.5 PCCS Tone Classification Algorithm

- **Skin Tone 검출을 위한 데이터 분석**

1. 피부톤 검출을 위한 다량의 인물사진 데이터 마련
   : 팬톤에서 제공한 1770장의 인물사진 크롤링
2. 전처리
   : OpenCV를 이용하여 인물 배치 및 데이터 전처리
3. Facial Landmark 추출
   : OpenCV Libarary를 이용하여 얼굴영역 구분 및 볼의 피부색 추출
4. Machine Learning Ensemble 기법을 이용하여 분류
   : 웜톤, 쿨톤 피부색의 팔레트와 비교하여 톤을 분류, Lab 중 a,b 값을 이용하여 앙상블로 결정 바운더리 분류

- **앙상블(Ensemble)**

  피부톤 데이터를 웜톤과 쿨톤으로 매칭하는 여러 모델(여러 결정트리)를 결합하여 하나의 결정트리를 생성하여 웜톤과 쿨톤을 예측하고 피부톤을 분류한다.

```
Warm=[[243, 129, 141], [238, 131, 140], [231, 134, 146], [224, 136, 151], [205,
```

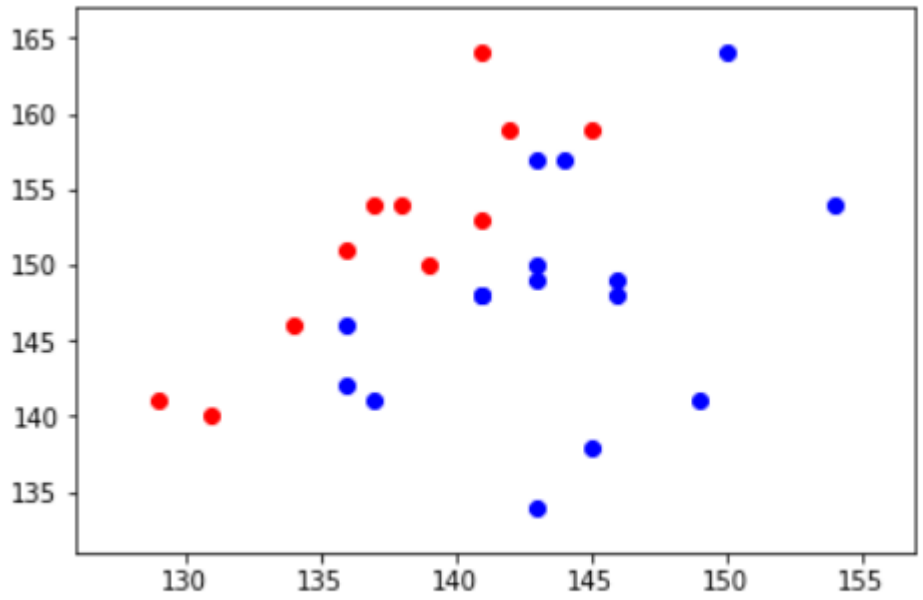
```
139, 150], [194, 138, 154], [190, 141, 164], [192, 141, 153], [175, 145, 159],
[184, 137, 154], [185, 142, 159]]

Cool=[[231, 136, 142], [228, 136, 146], [217, 143, 149], [214, 137, 141], [193,
146, 148], [184, 141, 148], [182, 143, 150], [183, 146, 149], [118, 150, 164],
[127, 143, 157], [112, 144, 157], [92, 141, 148], [64, 154, 154], [57, 145, 138],
[53, 149, 141], [51, 143, 134]]
```

```
[n[1]/n[0]-n[2]/n[0] for n in Warm]
```

```
[n[1]/n[0]-n[2]/n[0] for n in Cool]
```

## Decision

```
fig = plt.figure()
ax = fig.gca()
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
#z = np.linspace(0, 300, 100)

x_min = min([n[1] for n in Warm]+[n[1] for n in Cool]) - 3
x_max = max([n[1] for n in Warm]+[n[1] for n in Cool]) + 3

y_min = min([n[2] for n in Warm]+[n[2] for n in Cool]) - 3
y_max = max([n[2] for n in Warm]+[n[2] for n in Cool]) + 3

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)

ax.plot( [n[1] for n in Warm],[n[2] for n in Warm],'ro')
ax.plot( [n[1] for n in Cool],[n[2] for n in Cool],'bo')
```



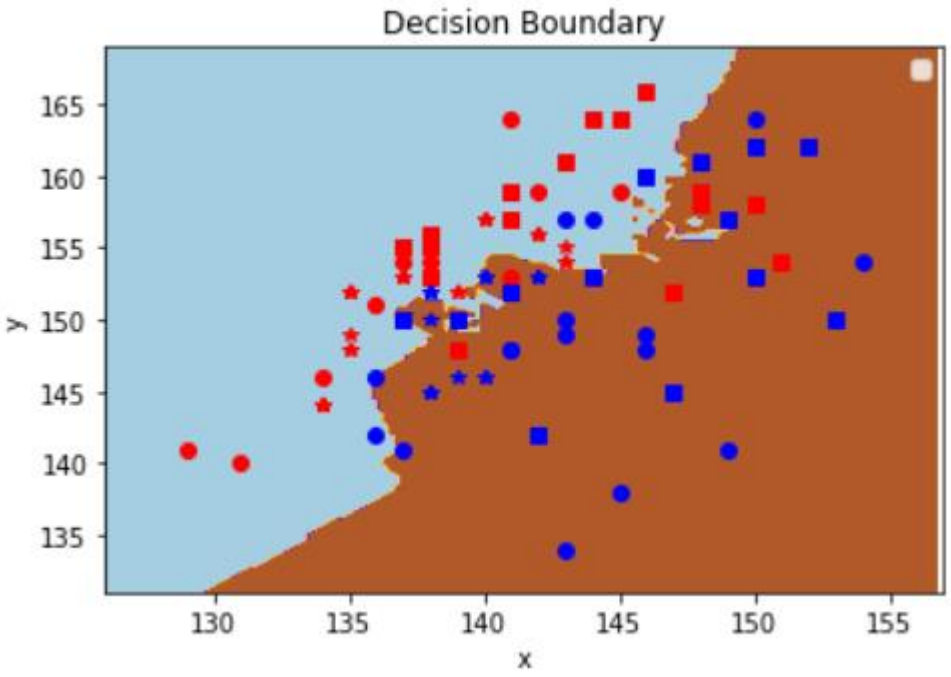

Red dots are Warm tone, Blue dots are Cool tone. X axis is A, Y axis is B.

## Ensemble

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from itertools import product
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB
#data = np.loadtxt("data.txt", delimiter=',')
#dim = warm.shape

#Warm = warm+warm2+warm4+warm5
#Cool = cool+cool2+cool4+warm5

Warm = warm +warm2+warm5
Cool = cool+cool2+cool5
x_min = min([n[0] for n in Warm]+[n[0] for n in Cool]) - 3
x_max = max([n[0] for n in Warm]+[n[0] for n in Cool]) + 3
y_min = min([n[1] for n in Warm]+[n[1] for n in Cool]) - 3
y_max = max([n[1] for n in Warm]+[n[1] for n in Cool]) + 3

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)


plt.plot( [n[0] for n in warm],[n[1] for n in warm],'ro')
plt.plot( [n[0] for n in cool],[n[1] for n in cool],'bo')

plt.plot( [n[0] for n in warm5],[n[1] for n in warm5],'ro', marker='s')
plt.plot( [n[0] for n in cool5],[n[1] for n in cool5],'bo', marker='s')

#plt.plot( [n[0] for n in warm3],[n[1] for n in warm3],'ro', marker='*')
#plt.plot( [n[0] for n in cool3],[n[1] for n in cool3],'bo', marker='*')

plt.plot( [n[0] for n in warm2],[n[1] for n in warm2],'ro', marker='*')
plt.plot( [n[0] for n in cool2],[n[1] for n in cool2],'bo', marker='*')


X = Warm+ Cool
Y = [0 for n in Warm] + [1 for n in Cool]

y = Y

bdt1 = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0)
bdt2 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), algorithm="SAMME",
n_estimators=200)
bdt3 = KNeighborsClassifier(n_neighbors=3)
bdt4 = KNeighborsClassifier(n_neighbors=9)
bdt5 = SVC(kernel='poly', probability=True)
bdt6 = GaussianNB()
bdt7 = KNeighborsClassifier(n_neighbors=5)


eclf = VotingClassifier(estimators=[('gbc', bdt1),('ada', bdt2),('dtc',
bdt3),('knn', bdt4),('svc', bdt5),('gnb', bdt6),('knn3', bdt7)], voting='soft',
\
                    weights=[0.01,0.01,0.5,2,0.5,2,0.5])
bdt = eclf
```

```
bdt1 = bdt1.fit(X,Y)
bdt2 = bdt2.fit(X,Y)
bdt3 = bdt3.fit(X,Y)
bdt4 = bdt4.fit(X,Y)
bdt5 = bdt5.fit(X,Y)
bdt6 = bdt6.fit(X,Y)
bdt7 = bdt7.fit(X,Y)

#clf1 = clf1.fit(X,y)
#clf2 = clf2.fit(X,y)
#clf3 = clf3.fit(X,y)
#bdt = eclf.fit(X,y)



#bdt = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
max_depth=1, random_state=0)
#bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), algorithm="SAMME",
n_estimators=200)
#bdt = DecisionTreeClassifier(max_depth=4)
#bdt = KNeighborsClassifier(n_neighbors=7)
#bdt = SVC(kernel='rbf', probability=True)
#bdt = SGDClassifier(loss="hinge", penalty="l2")#
#bdt = KNeighborsClassifier()
#bdt = GaussianNB()
bdt = bdt.fit(X, Y)

# ploting parameters setting
plot_colors = "rb"
plot_step = 0.2
class_names = "AB"

# plot the decision boundary
#x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
#y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
#x_min = y_min = 100
#x_max = y_max = 250

xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),np.arange(y_min, y_max,
plot_step))
Z = bdt.predict(np.c_[xx.ravel(), yy.ravel()])
print(Z)

Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)

# plot the training points
#for i, n, c in zip(range(2), class_names, plot_colors):
    #idx = np.where(Y == i)
#    plt.scatter(X[idx, 0], X[idx, 1], c=c, cmap=plt.cm.Paired, label="Class %s" %
n)

plt.legend(loc='upper right')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Boundary')

plt.show()
```




- **PCCS 톤 분류**

. PCCS의 각 영역의 모든 색의 S, V 중간 값 추출
(이때, PCCS 각 영역의 색은 12색을 기준으로 함)

| | x | y | Type |
|---|---|---|---|
| **0** | 0.874645 | 181.125000 | vivid |
| **1** | 0.692095 | 207.416667 | bright |
| **2** | 0.834804 | 170.166667 | strong |
| **3** | 0.890150 | 137.416667 | deep |
| **4** | 0.330006 | 221.166667 | light |
| **5** | 0.398006 | 182.000000 | soft |
| **6** | 0.500221 | 138.583333 | dull |
| **7** | 0.681574 | 93.583333 | dark |
| **8** | 0.110734 | 226.583333 | pale |
| **9** | 0.285509 | 110.250000 | grayish |
| **10** | 0.269131 | 58.750000 | dark_grayish |
| **11** | 0.131884 | 186.750000 | light_grayish |

```
//Vivid
['#b91f57', 185, 31, 87],
['#d02f48', 208, 47, 72],
['#dd443b', 221, 68, 59],
['#e95b23', 233, 91, 35],
['#eec900', 238, 201, 0],
['#00a15a', 0, 161, 90],
['#00926e', 0, 146, 110],
['#007488', 0, 116, 136],
['#00709b', 0, 112, 155],
```

```
['#005ba5', 0, 91, 165],
['#81378a', 129, 55, 138],
['#ad2e6c', 173, 46, 108]]

//Bright
['#ef6c70', 239, 108, 112],
['#fa8155', 250, 129, 85],
['#ffad36', 255, 173, 54],
['#fad831', 250, 216, 49],
['#b7c82b', 183, 200, 43],
['#41b879', 65, 184, 121],
['#00aa9f', 0, 170, 159],
['#0098b9', 0, 152, 185],
['#2981c0', 41, 129, 192],
['#7574bc', 117, 116, 188],
['#a165a8', 161, 101, 168],
['#d0678e', 208, 103, 142]]

//Strong
['#c53f4d', 197, 63, 77],
['#cc572e', 204, 87, 46],
['#e19215', 225, 146, 21],
['#debc03', 222, 188, 3],
['#9cad00', 156, 173, 0],
['#008f56', 0, 143, 86],
['#00827c', 0, 130, 124],
['#006f92', 0, 111, 146],
['#005b9b', 0, 91, 155],
['#534c98', 83, 76, 152],
['#7c3d84', 124, 61, 132],
['#a33c6a', 163, 60, 106]]

//Deep
['#a61d39', 166, 29, 57],
['#ab3d1d', 171, 61, 29],
['#b16c00', 177, 108, 0],
['#b39300', 179, 147, 0],
['#748400', 116, 132, 0],
['#007243', 0, 114, 67],
['#006664', 0, 102, 100],
['#005476', 0, 84, 118],
['#004280', 0, 66, 128],
['#3e337b', 62, 51, 123],
['#612469', 97, 36, 105],
['#861d55', 134, 29, 85]]

//Light
['#f6aba5', 246, 171, 165],
['#ffb99e', 255, 185, 158],
['#ffce90', 255, 206, 144],
['#fbe68f', 251, 230, 143],
['#d8df92', 216, 223, 146],
['#9cd9ac', 156, 217, 172],
['#7eccc1', 126, 204, 193],
['#79baca', 121, 186, 202],
['#83a7c8', 131, 167, 200],
['#a29fc7', 162, 159, 199],
['#b89ab8', 184, 154, 184],
['#daa0b3', 218, 160, 179]]


//Soft
```

```
['#ca8281', 202, 130, 129],
['#da927a', 218, 146, 122],
['#dba66b', 219, 166, 107],
['#d3bd6c', 211, 189, 108],
['#adb66b', 173, 182, 107],
['#76b18a', 118, 177, 138],
['#54a39b', 84, 163, 155],
['#5192a4', 81, 146, 164],
['#5d7ea0', 93, 126, 160],
['#7878a0', 120, 120, 160],
['#907194', 144, 113, 148],
['#b4788b', 180, 120, 139]]

//Dull
['#a35a5c', 163, 90, 92],
['#af6954', 175, 105, 84],
['#b37f46', 79, 127, 70],
['#ab9446', 171, 148, 70],
['#858f46', 133, 143, 70],
['#4f8766', 79, 135, 102],
['#2a7b76', 42, 123, 118],
['#246a7d', 36, 106, 125],
['#34597d', 52, 89, 125],
['#54527c', 84, 82, 124],
['#6c4a71', 108, 74, 113],
['#8b4f65', 139, 79, 101]]


//Dark
['#692934', 105, 41, 52],
['#75362a', 117, 54, 42],
['#794d1c', 121, 77, 28],
['#74601f', 116, 96, 31],
['#525b20', 82, 91, 32],
['#23523a', 35, 82, 58],
['#004746', 0, 71, 70],
['#004558', 0, 69, 88],
['#123452', 18, 52, 82],
['#322d51', 50, 45, 81],
['#432848', 67, 40, 72],
['#612d46', 97, 45, 70]]

//Pale
['#e7d5d4', 231, 213, 212],
['#e9d5cf', 233, 213, 207],
['#f6e3ce', 246, 227, 206],
['#efe6c6', 239, 230, 198],
['#e6e9c6', 230, 233, 198],
['#c4e0cb', 196, 224, 203],
['#bfe0d9', 191, 224, 217],
['#c6dde2', 198, 221, 226],
['#c2ccd5', 194, 204, 213],
['#c9cad5', 201, 202, 213],
['#d0c8d1', 208, 200, 209],
['#e4d5d9', 228, 213, 217]]

//Grayish
['#745c5c', 116, 92, 92],
['#755c57', 117, 92, 87],
['#806c5c', 128, 108, 92],
['#786f57', 120, 111, 87],
['#6e725a', 110, 114, 90],
```

```
['#53665a', 83, 102, 90],
['#4e6764', 78, 103, 100],
['#4f656c', 79, 101, 108],
['#4c5765', 76, 87, 101],
['#565566', 86, 85, 102],
['#605262', 96, 82, 98],
['#725c63', 114, 92, 9]]

//Dark grayish
['#3e2d30', 62, 45, 48],
['#3f2e2c', 63, 46, 44],
['#4a3c32', 74, 60, 50],
['#443e30', 68, 62, 48],
['#3d4033', 61, 64, 51],
['#2a342e', 42, 52, 46],
['#273434', 39, 52, 52],
['#273439', 39, 52, 57],
['#222933', 34, 41, 51],
['#292734', 41, 39, 52],
['#302531', 48, 37, 49],
['#3d2e34', 61, 46, 52]]

//Light grayish
['#c0abaa', 192, 171, 170],
['#c1aba5', 193, 171, 165],
['#cebba8', 206, 187, 168],
['#c6bea1', 198, 190, 161],
['#bdc1a2', 189, 193, 162],
['#9db6a5', 157, 182, 165],
['#98b6b1', 152, 182, 177],
['#9eb4b9', 158, 180, 185],
['#9ba5af', 155, 165, 175],
['#a2a2af', 162, 162, 175],
['#aba0ab', 171, 160, 171],
['#bdacb0', 189, 172, 176]]
```

## 5.6 Tone Ranking

- Tone Ranking

진단의 정확성을 높이기 위해 서로 다른 조명(실내 조명, 태양광, 어두운 조명 등)에서 5 개의 사진을 입력 받는다.

입력 사진 5 개의 사진에 이미지처리 과정을 일괄 적용시켜 정확성을 향상시켰으며 Best Tone, Good Tone, …, Bad Tone, Worst Tone 순으로 사용자 각각의 Tone 순위를 계산한다.

| Pale | Light | L_G | Soft | Dull | Greyish | D_G | Dark | Bright | Strong | Vivid | Deep |
|---|---|---|---|---|---|---|---|---|---|---|---|

Tone 배열은 이러한 순서로 fix 하고, 각 image 에서 결정한 tone 정보를 토대로 이 tone 배열을 결정한다.

결정 방법은 각 image 에서,

(12 - image 에서 나온 tone 배열에서의 index 값)를 위의 tone 배열의 각 tone 에 저장한다.

이러한 방식을 5 개의 image 에서 모두 적용한 후, sorting 을 통해 값이 큰 tone 일수록 사용자의 tone 과 더 근접한 tone 으로 판정하여 최종 tone 을 결정한다.

이 때, tone 결정뿐만 아니라 실제로 여러 image 에서 이미지 처리를 통해 비슷한 tone 이 나온다는 것을 알 수 있기도 하였다.

- Experiment Example

| User 1 | |
|---|---|
| Image 1 | Bright – Light – Pale - … |
| Image 2 | Bright – Light – Soft - … |
| Image 3 | Light – Soft – Bright - … |
| Image 4 | Vivid – Bright – Strong - … |
| Image 5 | Bright – Light – soft - … |

대부분의 경우 톤의 S, V 이 가까운 경우들로 측정이 되었으며 평균적으로 5 번 수행하였을 경우 3 번은 같은 결과를 나타내었다.

## 5.7 2-Way Recommendation

1. Best Tone 의 Item Recommendation

- 사용자의 Tone 정보를 받아 각 Tone 에 해당하는 Product DB 출력

2. 사용자가 고민중인 아이템 두 가지를 사진으로 입력하고 입력 색상 중 Tone 순위를 활용해 색상 Recommendation

- 입력한 사진 두 개를 각각 Image Processing 후 중앙 RGB 값 추출
- 사용자 tone 결정과 같은 방법으로 각 사진의 tone 측정
- 배열에 best tone 부터 worst tone 이 저장되어 있다는 것을 활용하여 index 값을 가중치로 활용하여 두 개의 사진에 score 매김

  ex)

| Image 1 | D-G | Strong | Vivid | Bright | Dark | Greyish | L-G | Soft | Pale | Deep | Light | Dull |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

  Image 1 의 각 tone 의 score: (12-배열 index 값)

| Pale | Light | L_G | Soft | Dull | Greyish | D_G | Dark | Bright | Strong | Vivid | Deep |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 12-8 | 12-10 | 12-6 | 12-7 | 12-11 | 12-5 | 12-0 | 12-4 | 12-3 | 12-1 | 12-2 | 12-9 |

- 이와 같은 방식으로 Image1 과 Image 2 의 tone 정보를 score 화하고,
  Score 화할 때 사용한 배열의 index 는 사용자의 tone ranking 순서대로 나열한다.
- Score 배열에서 0~4 번지의 합을 더해서 더 점수가 큰 것을 사용자의 tone 과 matching 된다고 결정한다.

# 6 Schedule

| | 프로젝트 일정 | 4월 | | | | 5월 | | | | 6월 | 역할 분담 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 완료 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1,2 | |
| 준비` | 주제 선정 | | | | | | | | | | 공통 |
| | 자료 조사 | | | | | | | | | | |
| | 작업환경 설정 | | | | | | | | | | |
| 조사 | Skin Tone 데이터 조사 | | | | | | | | | | 민경 |
| | Color Match 데이터 조사 | | | | | | | | | | |
| | 추천 알고리즘 조사 | | | | | | | | | | 민지 |
| 설계 | 데이터 베이스 설계 | | | | | | | | | | 민지 |
| | 자료 구조 설계 | | | | | | | | | | |
| | 알고리즘 선정 | | | | | | | | | | 민경 |
| 구현 | 진단 및 추천 알고리즘 구현 | | | | | | | | | | 공통 |
| 보완 | 시스템 일반화 | | | | | | | | | | 공통 |
| | 시스템 최적화 및 수정 | | | | | | | | | | |
| 평가 | α테스트 | | | | | | | | | | |
| | β 테스트 | | | | | | | | | | |
| | 시연 및 발표 | | | | | | | | | | |

| | 프로젝트 일정 | 8월 | | | | 9월 | | | | 10,11월 | 역할 분담 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1-4 | |
| 준비` | 진단 알고리즘 보완 | | | | | | | | | | 민경 |
| | 추천 알고리즘 보완 | | | | | | | | | | 민지 |
| 조사 | 모바일 개발 환경 설정 | | | | | | | | | | 공통 |
| | 데이터 연동 | | | | | | | | | | |
| 디자인 | UX/UI 설계 | | | | | | | | | | 민지 |
| | UX/UI 구현 | | | | | | | | | | |
| | 디자인 보완 | | | | | | | | | | 민경 |
| 구현 | 최종 구현 | | | | | | | | | | 공통 |
| 보완 | 오류 및 개선사항 선정 | | | | | | | | | | 공통 |
| | 시스템 최적화 및 수정 | | | | | | | | | | |
| 평가 | α테스트 | | | | | | | | | | |
| | β 테스트 | | | | | | | | | | |
| | 시연 및 발표 | | | | | | | | | | |

# 7 Conclusion

- Contribution

- 기존 정성적 Personal color 측정을 정량적 Tone 측정으로 변환
- 외부 환경 영향 최소화를 통한 이미지 색상 추출

- Expected Effect

- 아이템 추천 데이터들도 PCCS Tone Detection알고리즘으로 세부 분류, 구조화하고 다량의 데이터를 수집하여 데이터와 서버를 확장하여 배포한다.
- 판별된 스킨톤들로 데이터 셋을 만든 뒤 딥러닝 방법으로 정확도를 높인다.
- 조명에 따른 측정 결과의 실험을 좀 더 구체화하여 성능을 평가한다.

# 8 Reference

- 개발 환경

OS

- Windows 10

S/W

- Android Studio 3.1.4 with NDK
- OpenCV 3.4.2
- Android 6.0.1 API 23
- Android Vision API 9.1.0
- Python Package - Scipy, Tensorflow, Numpy

- 참고 자료


http://www.omelkoimaging.com/

https://www.spiedigitallibrary.org/journals/Journal-of-Electronic-Imaging/volume-25/issue-6/061618/Enhanced-codebook-algorithm-for-fast-moving-object-detection-from-dynamic/10.1117/1.JEI.25.6.061618.short?SSO=1

https://stackoverflow.com/questions/8884972/color-detection-opencv

http://mystes.net/it/imageprocessing/camera-white-balance-awb-image-processing/

http://www.easyrgb.com/en/math.php