Personal Color Classification

송태인, 김준호, 박성민, 서승수, 최영현, 한형진

목차

요약 | Overview

주제

퍼스널 컬러 분류

학습 방법

데이터 전처리
모델 선정
학습

결과

시각화
트러블 슈팅

1

퍼스널 컬러 분류

Personal Color Classification

Part 1

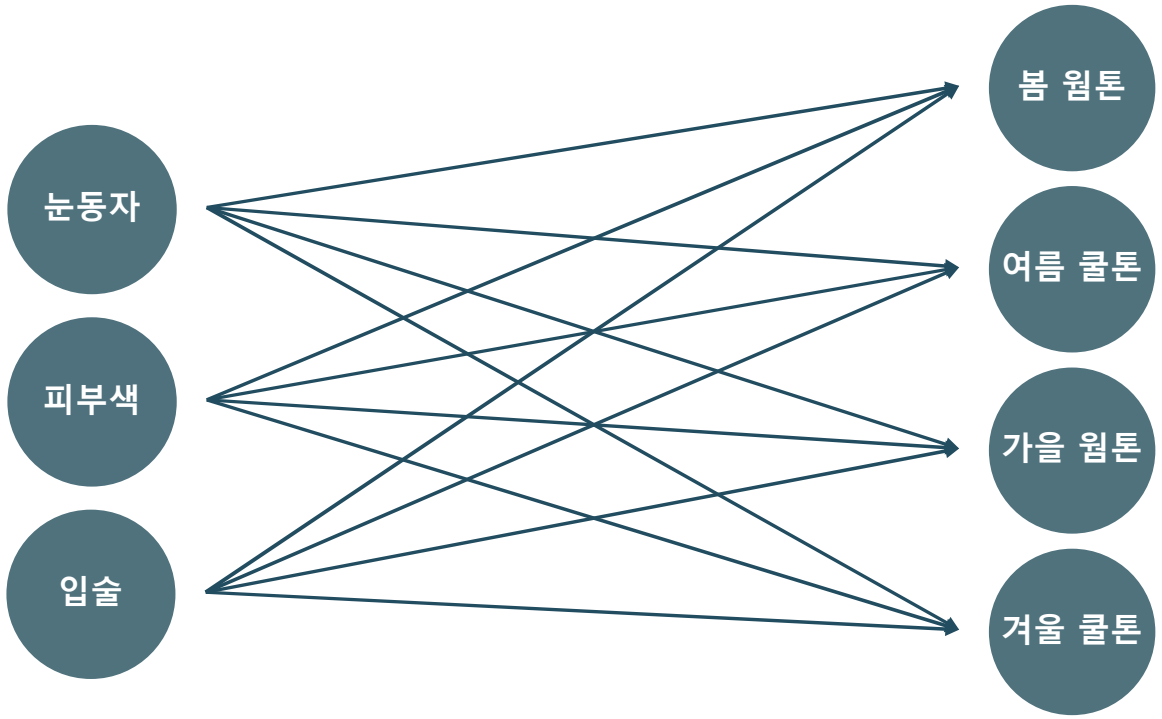
퍼스널 컬러란?

퍼스널 컬러(Personal Color)

사람의 얼굴에 가장 어울리는 색상을 찾는
미용 이론

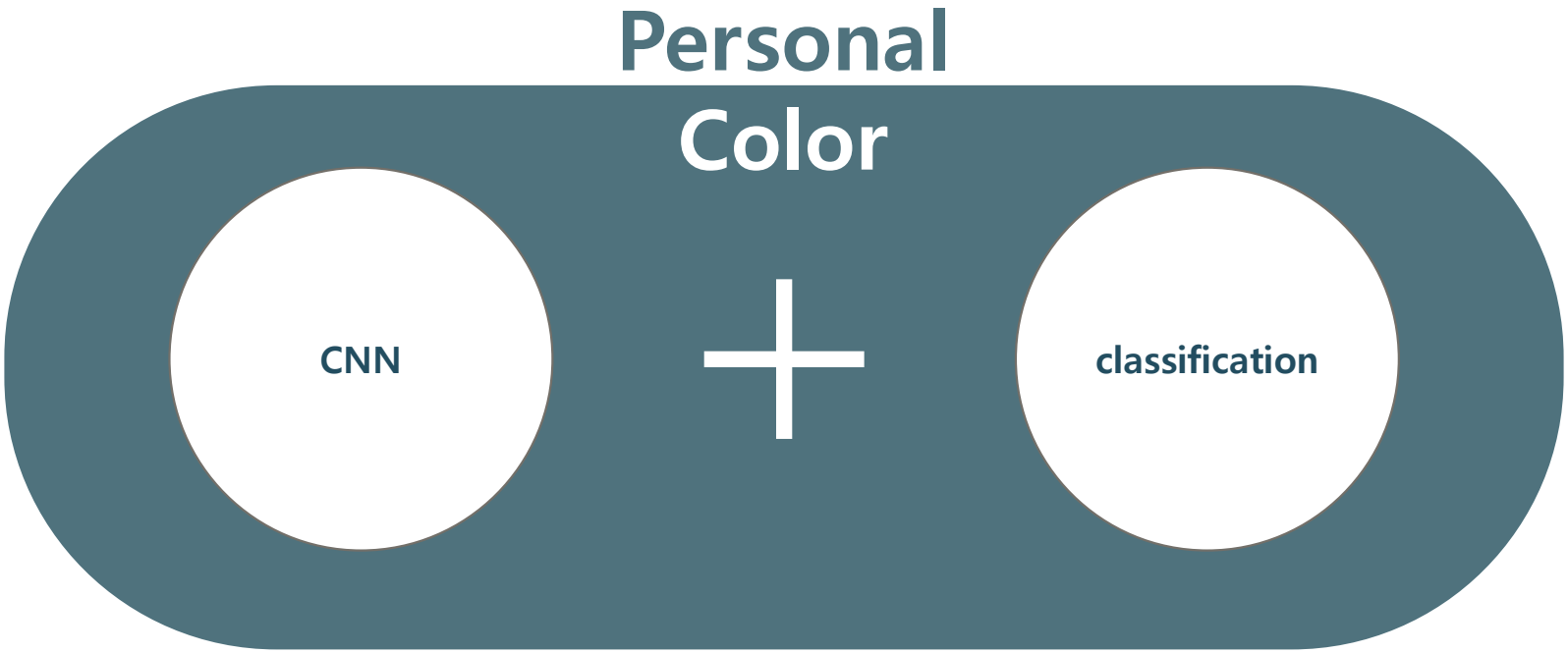
가장 어울리는 색상을
웜톤, 쿨톤이나 봄, 여름, 가을, 겨울로 부르는 것

주로 눈동자, 피부, 입술 색상으로 구분할 수 있음



Part 1

퍼스널 컬러 분류 방법



2

데이터 전처리

Part 2

전체 데이터 전처리

데이터 수집

1. 크롤링 진행

2. 계절별 퍼스널 컬러 연예인 사진 수집

>>

육안 검사

1. 흑백 및 채색이 짙은 이미지 제거

2. 동시에 두 명 이상의 인물이 나오는 이미지 제거

3. 마스크나 선글라스 착용한 이미지 제거

>>

이미지 자르기

얼굴 외 데이터 제거

얼굴 사이즈에 맞춰 이미지 자르기

Part 2

전체 데이터 전처리(데이터 수집)

Part 2

전체 데이터 전처리(육안 검사)

두 명 이상의 인물이 나오는 이미지

흑백 및 채색이 짙은 이미지

마스크나 선글라스 착용한 이미지

Part 2

전체 데이터 전처리(이미지 자르기)

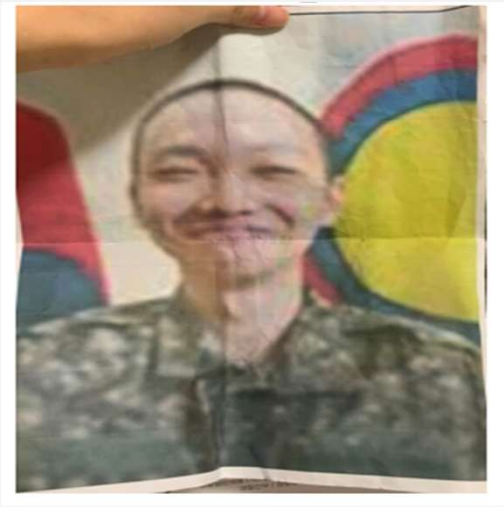
3

모델 선정 및 학습

Part 3

CNN

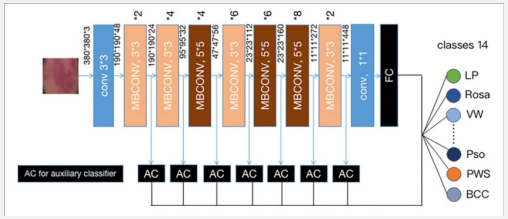
이미지 준비



>>

모델 학습

Efficientnet - b4



>>

결과 비교

CNN

```
# 이미지 증강
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor()
    ]),
    'validation': transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor()
    ]),
}
```

```
# 데이터셋 data_transforms 적용
image_datasets = {
    'train': datasets.ImageFolder('/content/drive/MyDrive/퍼스널컬러/train', data_transforms['train']),
    'validation': datasets.ImageFolder('/content/drive/MyDrive/퍼스널컬러/validation', data_transforms['validation'])
}
```

```
# 데이터로더
dataloaders = {
    'train': torch.utils.data.DataLoader(image_datasets['train'], batch_size=128, shuffle=True),
    'validation': torch.utils.data.DataLoader(image_datasets['validation'], batch_size=128, shuffle=False)
}
```

CNN

```
# 모델 생성
model = EfficientNet.from_pretrained('efficientnet-b4', num_classes=4).to(device)
print(model)
```

```
# 파라미터는 수정하지 않고 fc 모델만 수정(output이 4)
for param in model.parameters():
    param.requires_grad = False

# 레이어 쌓기
model._fc = nn.Sequential(
    nn.Linear(1792, 512),
    nn.ReLU(),
    nn.Linear(512, 4)
).to(device)
```

CNN

```python
# lr=0.00001: 10바퀴 학습
optimizer = optim.Adam(model.parameters(), lr=0.00001)
epochs = 20
for epoch in range(epochs):
    for phase in ['train', 'validation']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        sum_losses = 0
        sum_accs = 0
        for x_batch, y_batch in dataloaders[phase]:
            x_batch = x_batch.to(device)
            y_batch = y_batch.to(device)
            y_pred = model(x_batch)
            loss = nn.CrossEntropyLoss()(y_pred, y_batch)

            if phase == 'train':
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            sum_losses = sum_losses + loss.item()
            y_prob = nn.Softmax(dim=1)(y_pred)
            y_pred_index = torch.argmax(y_prob, axis=1)

            acc = (y_batch == y_pred_index).float().sum() / len(y_batch) * 100
            sum_accs = sum_accs + acc.item()

        avg_loss = sum_losses / len(dataloaders[phase])
        avg_acc = sum_accs / len(dataloaders[phase])
        print(f'{phase:10s}: Epoch {epoch+1:4d}/{epochs}, Loss: {avg_loss:.4f}, Accuracy: {avg_acc:.2f}%')
```

```
train     : Epoch   15/20, Loss: 1.2140, Accuracy: 45.00%
validation: Epoch   15/20, Loss: 1.2724, Accuracy: 41.02%
train     : Epoch   16/20, Loss: 1.2124, Accuracy: 44.57%
validation: Epoch   16/20, Loss: 1.2708, Accuracy: 42.05%
train     : Epoch   17/20, Loss: 1.2162, Accuracy: 45.76%
validation: Epoch   17/20, Loss: 1.2673, Accuracy: 41.38%
train     : Epoch   18/20, Loss: 1.2116, Accuracy: 44.37%
validation: Epoch   18/20, Loss: 1.2710, Accuracy: 41.58%
train     : Epoch   19/20, Loss: 1.2219, Accuracy: 44.31%
validation: Epoch   19/20, Loss: 1.2724, Accuracy: 42.63%
train     : Epoch   20/20, Loss: 1.2133, Accuracy: 43.83%
validation: Epoch   20/20, Loss: 1.2743, Accuracy: 40.28%
```

Loss : 1.2743, Accuracy : 40.28 %

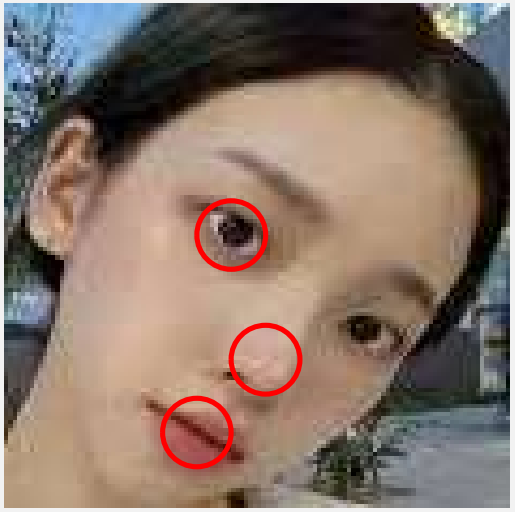
Part 3

Classification

이미지 준비

>>

얼굴 영역 선택 후 피부색 추출



>>

추출 값을 비교 후 퍼스널 컬러 선정

	eye_R	eye_G	eye_B	nose_R	nose_G	nose_B	mouse_R	mouse_G	mouse_B	tone
0	39	5	0	235	193	179	164	96	93	0
1	142	113	99	213	170	151	173	136	127	0
2	76	67	72	201	167	166	235	119	156	0
3	180	159	154	190	141	136	198	77	84	0
4	15	0	0	189	140	136	182	99	107	0
...	...	...	...	...	...	...	...	...	...	...
891	175	140	136	162	108	96	185	120	116	0
892	186	163	149	187	157	147	124	83	77	0
893	40	29	33	152	124	123	103	55	67	0
894	58	36	39	162	143	129	87	12	6	0
895	154	109	103	185	139	116	173	94	87	0

Part 3

Classification

	eye_R	eye_G	eye_B	nose_R	nose_G	nose_B	mouse_R	mouse_G	mouse_B	tone
0	39	5	0	235	193	179	164	96	93	0
1	142	113	99	213	170	151	173	136	127	0
2	76	67	72	201	167	166	235	119	156	0
3	180	159	154	190	141	136	198	77	84	0
4	15	0	0	189	140	136	182	99	107	0
...	...	...	...	...	...	...	...	...	...	...
4340	186	148	135	214	167	159	214	167	159	3
4341	145	103	79	220	160	132	220	160	132	3
4342	191	158	139	247	201	178	247	201	178	3
4343	78	38	30	224	172	158	224	172	158	3
4344	50	34	19	217	186	157	217	186	157	3

4345 rows × 10 columns

```
# 데이터 불러오기
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/9. 자연어처리/퍼스널컬러 서브 프로젝트/bio_data.csv')
df = df.drop('Unnamed: 0', axis = 1)

# 학습 - 테스트 분리
X_train, X_test, y_train, y_test = train_test_split(df.drop('tone', axis = 1),
                                                    df['tone'],
                                                    test_size = 0.2,
                                                    random_state = 17)
```

Part 3

Classification

1

```
# 모델 선정
logreg = LogisticRegression()

# 파라미터 선정
param_grid = {
    'C': [0.1, 1.0, 10.0],
    'penalty': ['l1', 'l2']
}
```

2

```
# 그리드 서치를 사용한 하이퍼파라미터 튜닝
grid_search = GridSearchCV(logreg, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# 최적의 모델로 예측 수행
best_model = grid_search.best_estimator_
pred = best_model.predict(X_test)
```

3

```
# 실제 값과 예측 값 비교 scatter plot
plt.scatter(range(len(y_test)), y_test, color='b', label='Actual')
plt.scatter(range(len(pred)), pred, color='r', label='Predicted')
plt.xlabel('Sample')
plt.ylabel('Class')
plt.title('Actual vs Predicted')
plt.legend()
```

4

결과

Part 4

결과

pred:winter_cool
Actual:summer_cool

pred:winter_cool
Actual:winter_cool

pred:summer_cool
Actual:summer_cool

pred:autumn_warm
Actual:winter_cool

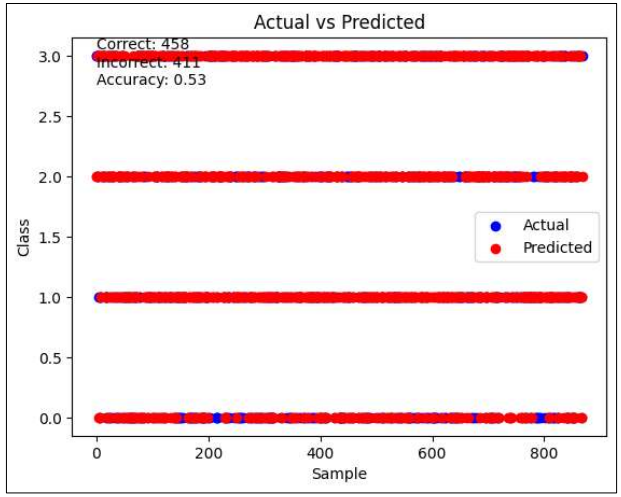
pred:spring_warm
Actual:spring_warm

pred:spring_warm
Actual:autumn_warm

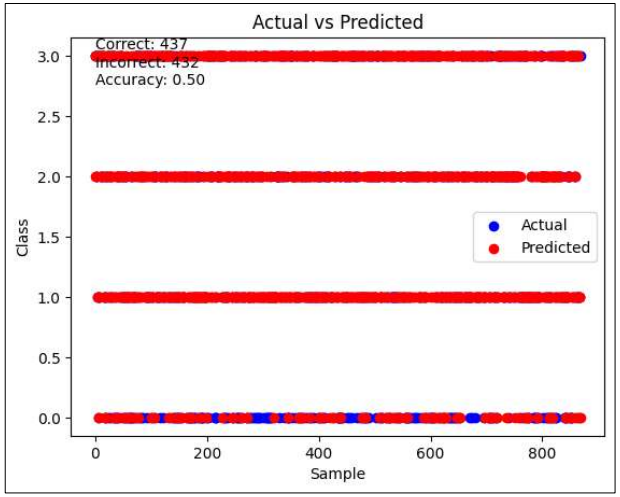
Part 4

결과

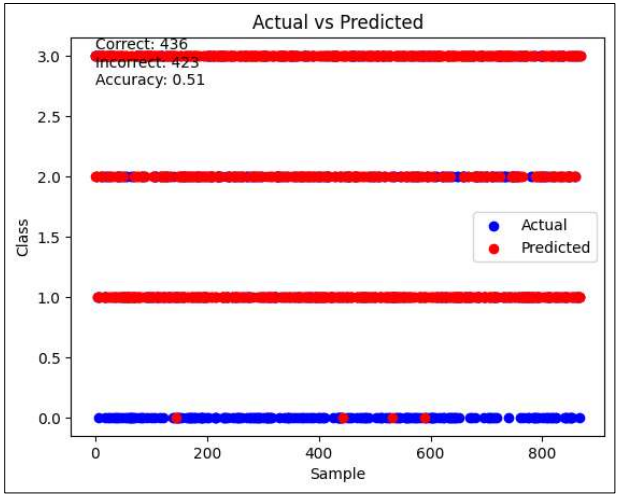
RandomForest



Logistic Regressor



SVC



Part 4

결과

항목	CNN	Classification
Data	Img	csv
len	4,303	4,345
Model	EfficientNet – b4	RandomForest Logistic Regressor SVC
Accuracy	약 40 ~ 45 %	약 50 %

Part 4

Trouble shooting

얼굴 영역 선택의 어려움

OpenCV, MidiaPipe를 이용하여 얼굴 영역만 정확하게 추출
CNN, Classification 학습에 용이하게 사용

```
import os
import cv2
import mediapipe as mp

# 이미지 파일 경로를 지정합니다.
folder_path = '/content/drive/MyDrive/퍼스널 컬러/연애인 테스트 사진' #해당 이미지 데이터셋 경로 설정
IMAGE_FILES = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]

# Mediapipe의 얼굴 감지 모델을 로드합니다.
mp_face_detection = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils
```

```
with mp_face_detection.FaceDetection(
    model_selection=1, min_detection_confidence=0.5) as face_detection:
  for idx, file in enumerate(IMAGE_FILES):
    image = cv2.imread(file)
    # 작업 전에 BGR 이미지를 RGB로 변환합니다.
    results = face_detection.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    # 이미지를 출력하고 그 위에 얼굴 박스를 그립니다.
    if not results.detections:
      continue
    annotated_image = image.copy()
    for detection in results.detections:
      mp_drawing.draw_detection(annotated_image, detection)
```