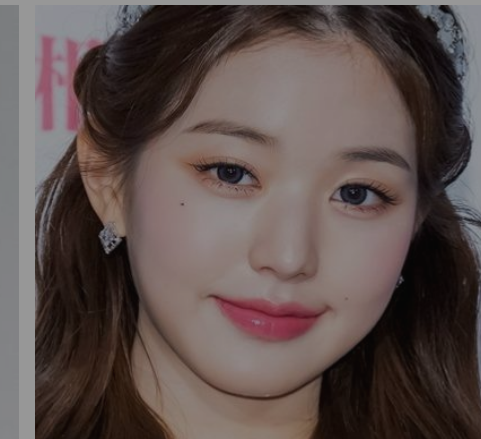
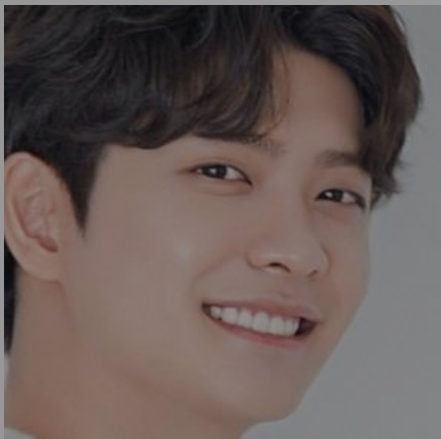


# Face Model

Age Idol Tone Harry



Face book 이재필 최성진 김찬수 강요셉



# CONTENT

**주제선택이유**

**마법의 모자**  
(Sorting Hat)

**쿨톤킴톤 구분**  
(Face Tone)

**얼굴나이 예측**  
(Face Age)

**아이돌상 예측**  
(Face Idol)

# 주제선정배경

## 인천일보

성형 공화국의 그늘  
"예쁘면 고시 3관왕 하는 것과 같아"  
"잘생기면 다 오빠야"

개인주의 성향이 강한 현대사회에서 타  
독각적인 파락이 필요해졌고, 이 때  
중시하는 현상이 두드러지게 했다는 가  
개 인 의 사 소 한 외 모 콤플렉스  
외모지상주의의 도래는 사회적  
경종을 울리고 있다.

## 인천일보

마스크 못 벗는 사회  
코로나 엔데믹 선언 후 외모컴플렉스로  
취업 장벽을 느끼는 사람들이 생겨남

외모를 통해 개인 간의 우열뿐 아니라 인생의 성공과  
실패까지도 결정할 수 있다고 생각하는 사회적 풍조로,  
무분별한 외모지상주의는 가치관 형성에도 부정적  
영향을 끼치고 있다.

## 미디어 오늘

류희림 방통심의위원장 저서 논란  
방송뿐만 아니라 신문기자들도  
는 필기 실력보다는 외모 위주로  
로이 늘고 있다"

에 등장하는 여기자들을 잘 보라.  
널리스트로서 자질이 더 중요한 기자  
상주의의 영향이 미치지 않을까 하는

# 주제선정배경



외모에 대한 관심 증대



얼굴로 할 수 있는  
종합서비스는 미비

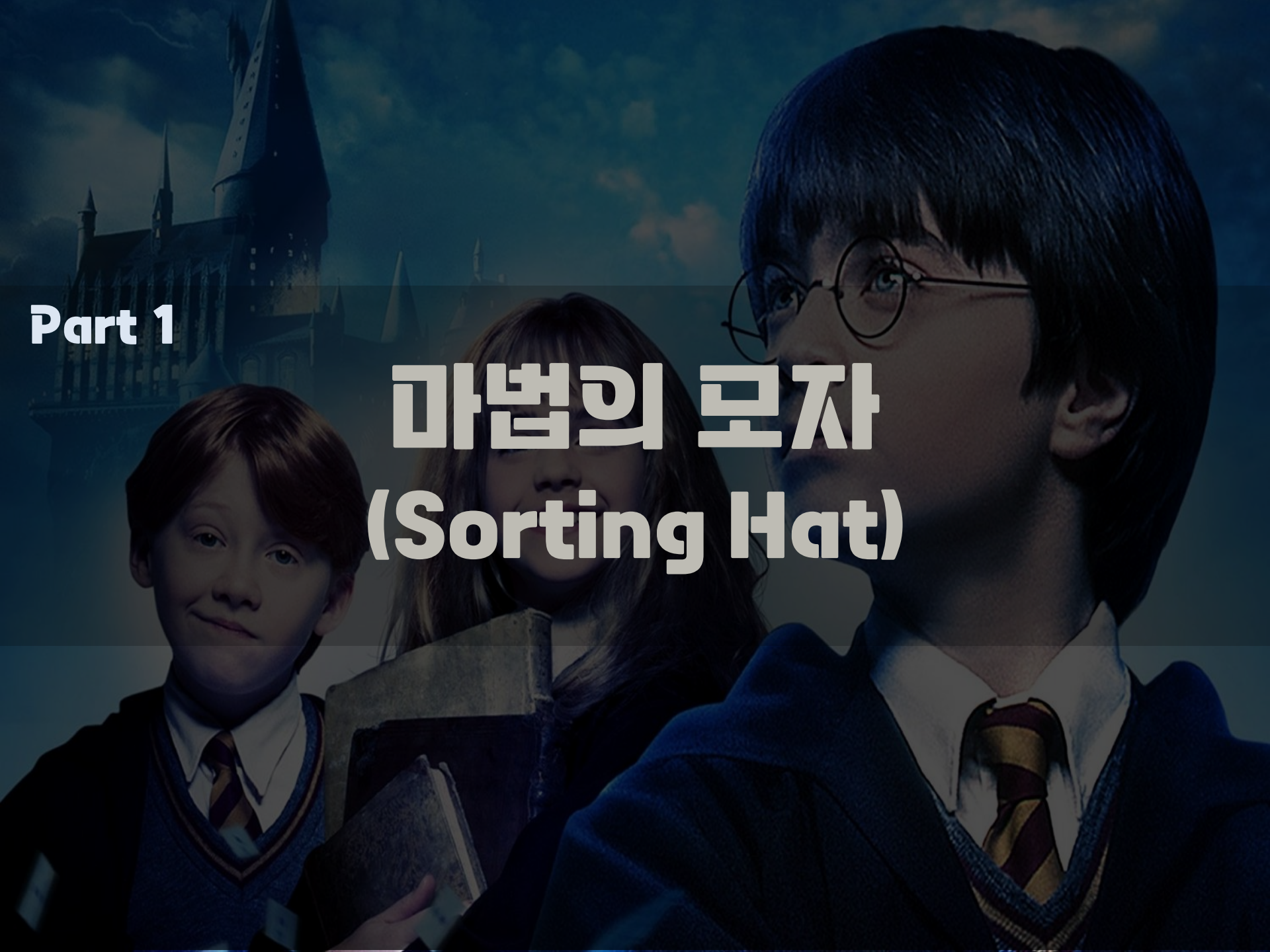


얼굴을 활용한  
다양한 서비스 구현



**Part 1**

# **마법의 모자 (Sorting Hat)**



# 1. Sorting Hat

"해리포터" 마법의 분류모자



〈해리포터〉의 마법학교 호  
그와트 신입생의  
기숙사를 분류해주는  
마법의 모자



"마법의 모자" 란?



# 1. Sorting Hat

"해리포터" 마법의 분류모자

\* 이미지 수집

\* 이미지 증강

\* 학습

\* 평가



주요과정



# 1. Sorting Hat

"해리포터" 마법의 분류모자

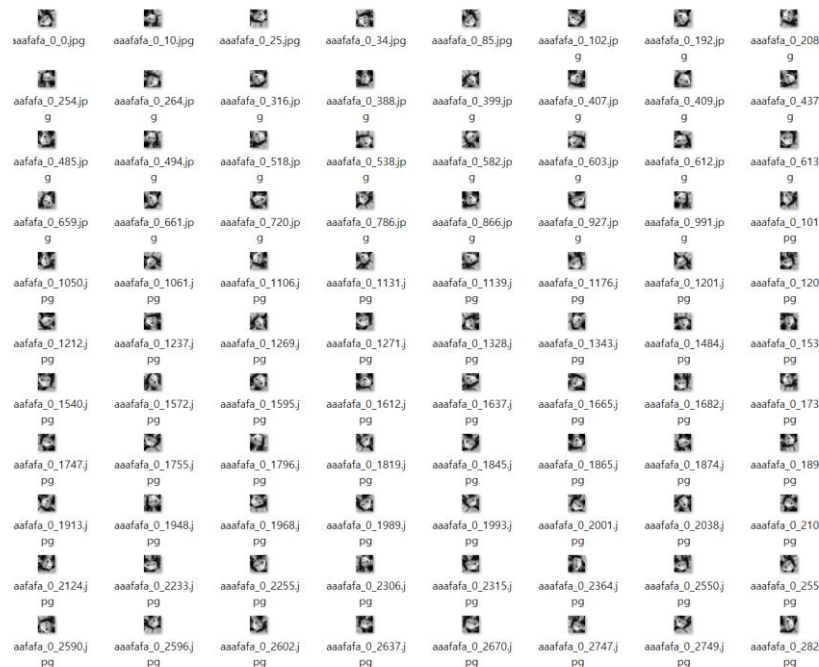
## 이미지 증강

ImageDataGenerator



코드분석

```
# ImageDataGenerator 인스턴스 생성
datagen = ImageDataGenerator(rescale=1/255.,
                              rotation_range=90,
                              width_shift_range=[-2, 2],
                              # height_shift_range=0.2,
                              # shear_range=0.25,
                              zoom_range=[0.8, 1.2],
                              horizontal_flip=True,
                              vertical_flip=True,
                              brightness_range=[0.2, 0.8],
                              fill_mode='nearest')
```





# 1. Sorting Hat

"해리포터" 마법의 분류모자

## 학습 및 결과

```
927/927 [=====] - 13s 1
Test Accuracy: 0.8016860485076904
```



결과분석

## CNN 기반

### AF

ELU, Leaky ReLU

출력층 - softmax

### 손실함수

sparse\_categorical\_crossentropy

### Optimizer

Nadam



결과분석

## Part 2

# 컬론 톤톤 구분 (Face Tone)

## 2. Face Tone

컬톤과 킴톤을 구별하는 모델 - 전처리



Imagetools 사이트에서  
수집한 이미지를 Face만 잘라내서 저장



이미지 배경제거와 그레이스케일 사용 X  
헤어 색과 얼굴 색감이 톤을 구별하기 때문



Pretreatment!



# 2. Face Tone

## 컬론과 톤톤을 구별하는 모델 - 전처리

```
# ImageDataGenerator를 사용하여 이미지 부풀리기 설정
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

**ImageDataGenerator**를  
사용하여 이미지를 증강시킴

💡 **Pretreatment!**



```
# 이미지를 부풀리고 저장
for i in range(2):
    file_list = os.listdir(folder_dir_ + img_dirs[i])
    for file_name in file_list:
        if file_name.endswith('.png') or file_name.endswith('.jpg') or file_name.endswith('.jpeg'):
            image_path = os.path.join(folder_dir_ + img_dirs[i], file_name)
            image = cv2.imread(image_path)
            rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            resized_image = cv2.resize(rgb_image, (img_width, img_height))
            x = resized_image.reshape((1,) + resized_image.shape)

            # flow() 메서드를 이용하여 부풀리기된 이미지를 생성
            for j, batch in enumerate(datagen.flow(x,
                                                    batch_size=1,
                                                    save_to_dir=gen_dir[i],
                                                    save_prefix='aug',
                                                    save_format='jpg')):
                if j > 999: # 각 원본 이미지당 새로운 이미지를 생성하는 개수 설정
                    break
```

**BGR로 저장되는 이미지를 RGB형태로 저장**  
**Flow 메서드**를 이용하여 이미지 생성



# 2. Face Tone

## 컬론과 웹톤을 구별하는 모델 - 모델 학습 방법

모델구성 : VGG 모델 기반

모델 컴파일

학습

모델 평가 및 저장

Flatten 레이어와 Dense 레이어를 쌓아  
최종적인 분류 모델 구성

손실함수 : categorical cross-entropy  
최적화 알고리즘 : Adam

Train\_generator 데이터를 가져와  
Validation\_generator를 통해 검증

Model.evaluate로 정확도와 손실 평가



Model Learning



## 2. Face Tone

### 컬론과 퓌톤을 구별하는 모델 - 모델 학습 방법

```
Train Loss: 0.31011253595352173
Train Accuracy: 0.871999979019165
8/8 [=====] -
Validation Loss: 0.5045158267021179
Validation Accuracy: 0.7914893627166748
```

정확도가 87%로 괜찮은 성능지표

But, Validation 정확도가 79%로 과적합

```
# 변수 지정
img_width, img_height = 32, 32
num_epochs = 8
batch_size = 32
num_classes = 2
```

이미지 크기 : 32 \* 32

에포크 수 : 8

배치사이즈 : 32

이미지 클래스 수 : 이진분류

 Model Learning



## Part 3

# 얼굴 나이 예측 (Face Age)



11101010100010101010101110011  
000101010001010101110010101001110101011  
11010101000101010101010111001  
10101010111001



NO: ONE PERSON  
GENDER: MAN  
AGE GROUP: YOUNG MAN  
ETHNICITY: CAUCASIAN  
HUMAN BODY PART: HUMAN FACE  
TIME: 167 S  
DETECTION: 63621 POINTS  
POS (X/Y/Z): 1322 / 856 / 21

# 3. Face Age

얼굴 나이 예측 서비스



얼굴사진을 통해  
미성년층, 청년층,  
장년층, 노년층을  
구분하는 서비스

내 얼굴 나이는?





# 3. Face Age

얼굴 나이 예측 서비스

1. 이미지 수집
2. 이미지 정제 및 증강
3. 학습 및 평가

💡 주요 과정



# 3. Face Age

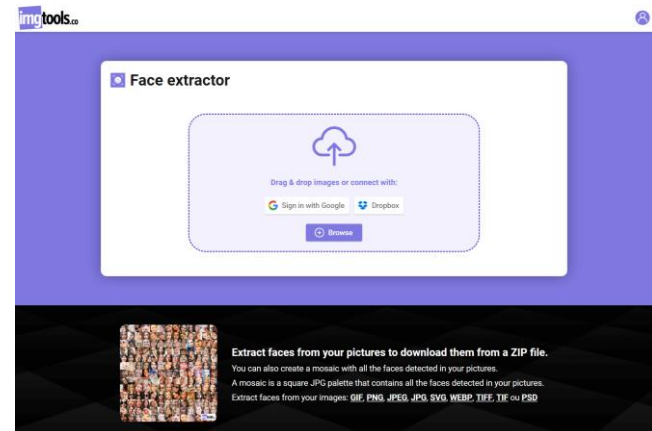
## 얼굴 나이 예측 서비스

### 1. 이미지 수집

💡 주요 과정

1) 미성년층, 청년층, 장년층, 노년층 각각 100장의 이미지를 가져옴.

2) 이미지 처리 서비스로 사람 머리만 정제. 하지만, opencv에 이미 있는 기능



<https://www.imgtools.co/face-extractor>

# 3. Face Age

얼굴 나이 예측 서비스

## 2. 이미지 증강

 주요 과정

### 1) ImageDataGenerator로 이미지 증강

```
# ImageDataGenerator 인스턴스 생성
datagen = ImageDataGenerator(rescale=1./255,
                              rotation_range=10,
                              width_shift_range=0.2,
                              height_shift_range=0.2,
                              shear_range=0.2,
                              zoom_range=[0.2, 2.2],
                              horizontal_flip=True,
                              vertical_flip=True,
                              fill_mode='nearest')
```

### 2) 이미지 개당 200개의 이미지 증강 뒤 리스트에 저장

```
total = []
for x in range(len(imgdatas)):
    x = imgdatas[x].reshape((1,) + imgdatas[x].shape)
    j = 0
    for batch in datagen.flow(x=x, batch_size=1, shuffle=False):
        total.append(batch)
        j += 1

    if j > 199: # 각 원본 이미지당 새로운 이미지를 생성하는 개수 설정
        break
```

✓ 36.2s

# 3. Face Age

## 얼굴 나이 예측 서비스

### 3. 학습 및 평가



1) 첫 학습 : CNN으로 진행. Conv2D 2개, 은닉층 2개로 구성

```
model = Sequential()
# model.add(transfer_model)

model.add(Conv2D(10,3,input_shape=(64,64,3),activation="relu",kernel_regularizer="l1",kernel_initializer="he_normal"))
model.add(MaxPool2D())
model.add(Conv2D(10,3,input_shape=(64,64,3),activation="relu",kernel_regularizer="l1",kernel_initializer="he_normal"))
model.add(MaxPool2D())

model.add(Flatten())
model.add(Dense(100,activation="relu",kernel_regularizer="l2",kernel_initializer="he_normal"))
model.add(Dense(100,activation="relu",kernel_regularizer="l2",kernel_initializer="he_normal"))
# model.add(Dense(100,activation="relu",kernel_regularizer="l2",kernel_initializer="he_normal"))
model.add(Dense(6,activation="softmax"))
```

2) Accuracy 0.16으로 나옴

```
model.fit(x=trainx,y=trainy,epochs=50,callbacks=[mc,es,lr],validation_split=0.2)

Epoch 1/50
1200/1200 [=====] - ETA: 0s - loss: 2.1979 - accuracy: 0.1707INFO:tensorflow:Assets written to: ./models\best_model\assets
INFO:tensorflow:Assets written to: ./models\best_model\assets
1200/1200 [=====] - 93s 76ms/step - loss: 2.1979 - accuracy: 0.1707 - val_loss: 1.8017 - val_accuracy: 0.1675 - lr: 0.0010
Epoch 2/50
1200/1200 [=====] - ETA: 0s - loss: 1.7946 - accuracy: 0.1653INFO:tensorflow:Assets written to: ./models\best_model\assets
INFO:tensorflow:Assets written to: ./models\best_model\assets
1200/1200 [=====] - 156s 138ms/step - loss: 1.7946 - accuracy: 0.1653 - val_loss: 1.7933 - val_accuracy: 0.1675 - lr: 0.0010
Epoch 3/50
245/1200 [=====>.....] - ETA: 1:38 - loss: 1.7931 - accuracy: 0.1673
```

-> 실패요인 분석 : 첫 시도 당시 10~60대  
각각 범주화하면서 특징을 잡지 못한 것으로 추정



# 3. Face Age

## 얼굴 나이 예측 서비스

### 3. 학습 및 평가



1) 두번째 학습 : 사전학습된 VGG16으로 진행.  
은닉층 1개로 구성

```
model1 = Sequential()
model1.add(transfer_model)

# model1.add(Conv2D(10,3,input_shape=(64,64,3),activation="relu",kernel_regularizer="l1",kernel_initializer="he_normal"))
# model1.add(MaxPool2D())
# model1.add(Conv2D(10,3,input_shape=(64,64,3),activation="relu",kernel_regularizer="l1",kernel_initializer="he_normal"))
# model1.add(MaxPool2D())

model1.add(Flatten())
model1.add(Dense(100,activation="relu",kernel_regularizer="l2",kernel_initializer="he_normal"))
# model1.add(Dense(100,activation="relu",kernel_regularizer="l2",kernel_initializer="he_normal"))
# model1.add(Dense(100,activation="relu",kernel_regularizer="l2",kernel_initializer="he_normal"))
model1.add(Dense(4,activation="softmax"))
```

2) Accuracy 0.55, 학습시간 269분

```
>>> model1.fit(x=train_x,y=train_y,epochs=50,callbacks=[mc,es,lr],validation_split=0.2)
[0] ✓ 200m 11.7s Python

Epoch 1/50
1600/1600 [=====] - ETA: 0s - loss: 1.2877 - accuracy: 0.5029INFO:tensorflow:Assets written to: ./models\best_model\assets
INFO:tensorflow:Assets written to: ./models\best_model\assets
1600/1600 [=====] - 827s 516ms/step - loss: 1.2877 - accuracy: 0.5029 - val_loss: 1.1808 - val_accuracy: 0.5084 - lr: 0.0010
Epoch 2/50
1600/1600 [=====] - ETA: 0s - loss: 1.1668 - accuracy: 0.5292INFO:tensorflow:Assets written to: ./models\best_model\assets
INFO:tensorflow:Assets written to: ./models\best_model\assets
1600/1600 [=====] - 761s 476ms/step - loss: 1.1668 - accuracy: 0.5292 - val_loss: 1.1450 - val_accuracy: 0.5384 - lr: 0.0010
Epoch 3/50
1600/1600 [=====] - ETA: 0s - loss: 1.1381 - accuracy: 0.5428INFO:tensorflow:Assets written to: ./models\best_model\assets
INFO:tensorflow:Assets written to: ./models\best_model\assets
1600/1600 [=====] - 748s 468ms/step - loss: 1.1381 - accuracy: 0.5428 - val_loss: 1.1195 - val_accuracy: 0.5441 - lr: 0.0010
Epoch 4/50
1600/1600 [=====] - ETA: 0s - loss: 1.1237 - accuracy: 0.5518INFO:tensorflow:Assets written to: ./models\best_model\assets
INFO:tensorflow:Assets written to: ./models\best_model\assets
```

3) 더 좋은 모델을 만들고 싶지만 학습시간이  
오래 걸리는 제한사항이 있어 Colab 사용

# 3. Face Age

## 얼굴 나이 예측 서비스

### 3. 학습 및 평가



1) 세번째 학습 : 사전학습된 VGG16으로 진행.  
은닉층 1-3개로 구성

```
+ 코드 + 텍스트
D. 1000/1000 [====...] - 25s 11s/step - loss: 0.7648 - accuracy: 0.7145 - val_loss: 0.8884 - val_accuracy: 0.6576 - lr: 1.0000e-04
Epoch 30/50
1000/1000 [====...] - 35s 22s/step - loss: 0.7618 - accuracy: 0.7160 - val_loss: 0.8864 - val_accuracy: 0.6607 - lr: 1.0000e-04
Epoch 35/50
1000/1000 [====...] - 25s 11s/step - loss: 0.7590 - accuracy: 0.7180 - val_loss: 0.8873 - val_accuracy: 0.6614 - lr: 1.0000e-04
Epoch 40/50
1000/1000 [====...] - 34s 21s/step - loss: 0.7545 - accuracy: 0.7193 - val_loss: 0.8808 - val_accuracy: 0.6682 - lr: 1.0000e-04
Epoch 45/50
1000/1000 [====...] - 33s 21s/step - loss: 0.7511 - accuracy: 0.7198 - val_loss: 0.8818 - val_accuracy: 0.6614 - lr: 1.0000e-04
Epoch 47/50
1000/1000 [====...] - 33s 11s/step - loss: 0.7481 - accuracy: 0.7214 - val_loss: 0.8767 - val_accuracy: 0.6623 - lr: 1.0000e-04
Epoch 48/50
1000/1000 [====...] - 25s 11s/step - loss: 0.7450 - accuracy: 0.7204 - val_loss: 0.8811 - val_accuracy: 0.6654 - lr: 1.0000e-04
Epoch 49/50
1000/1000 [====...] - 33s 11s/step - loss: 0.7417 - accuracy: 0.7204 - val_loss: 0.8803 - val_accuracy: 0.6641 - lr: 1.0000e-04
Epoch 50/50
1000/1000 [====...] - 33s 21s/step - loss: 0.7383 - accuracy: 0.7261 - val_loss: 0.8753 - val_accuracy: 0.6630 - lr: 1.0000e-04
Epoch 46/50
1000/1000 [====...] - 25s 11s/step - loss: 0.7364 - accuracy: 0.7279 - val_loss: 0.8766 - val_accuracy: 0.6630 - lr: 1.0000e-04
Epoch 47/50
1000/1000 [====...] - 27s 17s/step - loss: 0.7318 - accuracy: 0.7284 - val_loss: 0.8772 - val_accuracy: 0.6635 - lr: 1.0000e-04
Epoch 48/50
1000/1000 [====...] - 25s 11s/step - loss: 0.7308 - accuracy: 0.7284 - val_loss: 0.8785 - val_accuracy: 0.6655 - lr: 1.0000e-04
Epoch 49/50
1000/1000 [====...] - 25s 11s/step - loss: 0.7289 - accuracy: 0.7290 - val_loss: 0.8740 - val_accuracy: 0.6680 - lr: 1.0000e-04
Epoch 50/50
1000/1000 [====...] - 34s 21s/step - loss: 0.7248 - accuracy: 0.7308 - val_loss: 0.8708 - val_accuracy: 0.6694 - lr: 1.0000e-04
keras.src.callbacks.History at 0x784a4147c0b0

[10] model.evaluate(x=test_x, y=test_y)
500/500 [====...] - 7s 13s/step - loss: 0.8813 - accuracy: 0.6584
(0.8513948364227, 0.688979247595322)
```

2) Accuracy 0.65, 학습시간 25분.  
10배 이상 빨라짐.

3) 다양한 규제 및 은닉층 조절을 통해  
Accuracy 0.7 까지 올렸지만, 한계를 느낌

# 3. Face Age

## 얼굴 나이 예측 서비스

### 3. 학습 및 평가



#### 주요 과정

#### 1) 네번째 학습 : auto keras

```
import autokeras as ak

# 이미지 분류기 생성
clf = ak.ImageClassifier(max_trials=1) # max_trials는 시도할 모델의 수를 정의합니다.

# 데이터 로드 및 전처리

# 모델 학습
clf.fit(trainx, trainy, epochs=1000)

# 성능 평가
print('Accuracy: {accuracy}'.format(accuracy=clf.evaluate(testx, testy)))

... Using TensorFlow backend
```

#### 2) Accuracy 0.85 이상 올랐음. 하지만, RAM 부족으로 초기화됨.

```
1600/1600 [=====] - 7s 4ms/step - loss: 0.3793 - accuracy: 0.8540 - val_loss: 0.3544 - val_accuracy: 0.8666
Epoch 46/1000
1600/1600 [=====] - 7s 4ms/step - loss: 0.3810 - accuracy: 0.8545 - val_loss: 0.3461 - val_accuracy: 0.8701
Epoch 47/1000
1600/1600 [=====] - 7s 4ms/step - loss: 0.3684 - accuracy: 0.8696 - val_loss: 0.3607 - val_accuracy: 0.8636
Epoch 48/1000
1600/1600 [=====] - 7s 4ms/step - loss: 0.3761 - accuracy: 0.8576 - val_loss: 0.3422 - val_accuracy: 0.8730
Epoch 49/1000
1600/1600 [=====] - 7s 4ms/step - loss: 0.3725 - accuracy: 0.8574 - val_loss: 0.3507 - val_accuracy: 0.8697
Epoch 50/1000
1599/1600 [=====>.] - ETA: 0s - loss: 0.3708 - accuracy: 0.8582
```

# 3. Face Age

## 얼굴 나이 예측 서비스

### 3. 학습 및 평가

 주요 과정

#### 1) 다섯번째 학습 : auto keras Colab Pro 구매

Pay As You Go	Colab Pro	Colab Pro+	Colab Enterprise
<p>Pay As You Go</p> <p>\$9.99/100컴퓨팅 단위</p> <p>\$49.99/500컴퓨팅 단위</p> <p>현재 컴퓨팅 단위가 87.4% 있습니다.</p> <p>컴퓨팅 단위는 90일 후 만료됩니다. 필요한 양의 구매하십시오.</p> <ul style="list-style-type: none"> <li>✓ 구독이 필요하지 않습니다.</li> <li>✓ 사용량에 따른 비용을 지불하십시오.</li> <li>✓ 더 빠른 GPU</li> <li>✓ 더 강력한 GPU로 업그레이드하십시오.</li> </ul>	<p>Colab Pro</p> <p>월 \$9.99</p> <p>구매 요금제</p> <ul style="list-style-type: none"> <li>✓ 100컴퓨팅 단위/월</li> <li>✓ 컴퓨팅 단위는 90일 후 만료됩니다. 필요한 양의 구매하십시오.</li> <li>✓ 더 빠른 GPU</li> <li>✓ 더 강력한 GPU로 업그레이드하십시오.</li> <li>✓ 추가 메모리</li> <li>✓ 고성능 메모리 및 최신의 액세스하십시오.</li> <li>✓ 티라닐</li> <li>✓ 만일 VM으로 티라닐 사용 가능</li> <li>✓ 일부 국가 및 8세 이상만 해당</li> <li>✓ AI 지능 자동 완성</li> <li>✓ 입력 영역 미리 볼을 제공하는 지능형 주연이 자동으로 업데이트됩니다.</li> <li>✓ 코드 생성</li> <li>✓ 통합 지능을 비롯하여 자연어로 코드를 생성합니다.</li> </ul>	<p>Colab Pro+</p> <p>월 \$49.99</p> <p>Pro의 모든 혜택 및 다음 추가 혜택:</p> <ul style="list-style-type: none"> <li>✓ 추가 컴퓨팅 단위 400개가 제공되며 매월 총 500개 이상 가능</li> <li>✓ 컴퓨팅 단위는 90일 후 만료됩니다. 필요한 양의 구매하십시오.</li> <li>✓ 더 빠른 GPU</li> <li>✓ 더 강력한 프리미엄 GPU로 한발 앞서 업그레이드하십시오.</li> <li>✓ 백그라운드 실행</li> <li>✓ 컴퓨팅 단위를 통해 노드유저를 할라도 현재 실행 중인 노드백이 최대 24시간 동안 계속 실행됩니다.</li> </ul>	<p>Colab Enterprise</p> <p>사용량에 따른 비용 지불</p> <ul style="list-style-type: none"> <li>✓ 통합</li> <li>✓ BigQuery 및 Vertex AI와 같은 Google Cloud 서비스와 긴밀하게 통합됩니다.</li> <li>✓ Enterprise 노드백 스토리지</li> <li>✓ Google Drive 노드백 사용자 Cloud 콘솔 내에서 저장 및 공유하는 GCP 노드백으로 대역폭이 감소합니다.</li> <li>✓ 생산성</li> <li>✓ 생성형 AI 기반 코드 완성 및 생성</li> </ul>

거래금액
-12,986원

2) Accuracy 0.9 이상 올랐음.  
다만 실제 데이터에 판별 능력이 떨어짐.

-> 실패요인 분석 : 시도 당시 배경제거를  
하지 않고 진행



# 3. Face Age

## 얼굴 나이 예측 서비스

### 3. 학습 및 평가



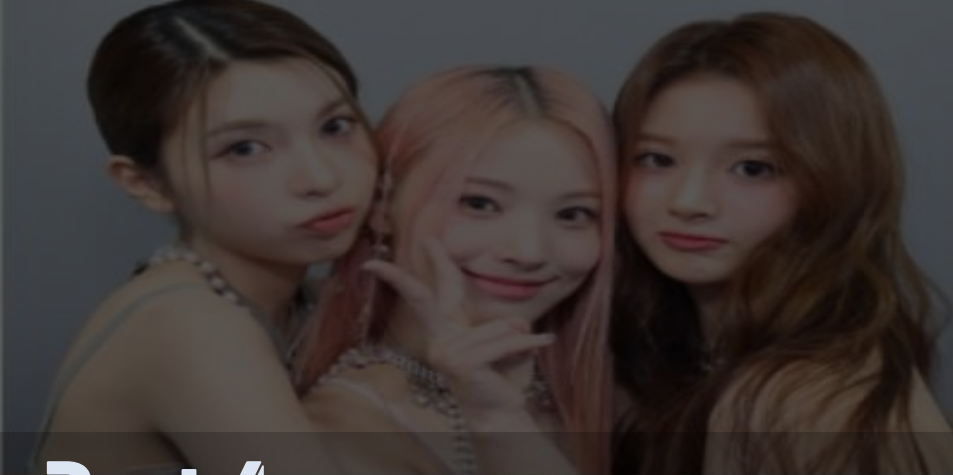
1) 마지막 학습 : auto keras

2) 배경제거를 하면서 Accuracy 0.86 으로 떨어졌지만, 실제 데이터에 판별 능력이 오름.

-> 결론 : 나이에 분류하기 위해 auto keras를 사용. 데이터의 영향을 많이 받는 특성들을 발견.

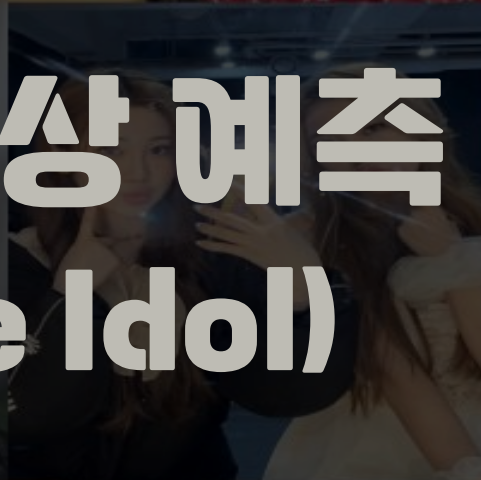
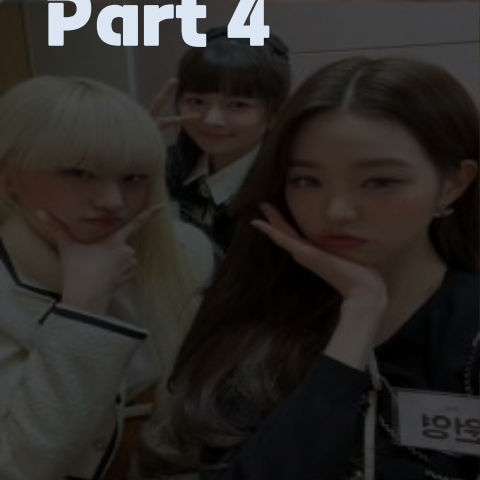
VGG-face, FaceNet, GoogLeNet 과 같은 전이학습 기법이 많이 활용됨.

추후, 분류 모델이 아닌 회귀 모델로 진행할 예정



**Part 4**

# 아이돌상 예측 (Face Idol)



# 4. Face Idol

내 얼굴형에 맞는 아이돌 그룹 찾기

- Aespa
- BlackPink
- 아이들
- Itzy
- Ive
- LeSserafim
- NewJeans



**NATURAL!**





# 4. Face Idol

학습 진행 과정



과정

i. 이미지 데이터 수집



ii. 이미지 데이터 전처리



iii. 데이터 증폭



iv. 학습



v. 평가

# 4. Face Idol

## 데이터 전처리



```
one_hot_labels = pd.get_dummies(labels, columns=[class_label])  
labels = one_hot_labels.values
```

# 각 클래스 라벨을 정수로 매핑하는 딕셔너리 생성

```
class_to_int_mapping = {  
    'Aespa': 0,  
    'BlackPink': 1,  
    'Chid': 2,  
    'Itzy': 3,  
    'IVE': 4,  
    'LeSserafim': 5,  
    'NewJeans': 6  
}
```

# 각 이미지의 라벨을 정수로 변환

```
ap_labels_int = [class_to_int_mapping['Aespa']] * len(ap_labels)  
bp_labels_int = [class_to_int_mapping['BlackPink']] * len(bp_labels)  
cd_labels_int = [class_to_int_mapping['Chid']] * len(cd_labels)  
iz_labels_int = [class_to_int_mapping['Itzy']] * len(iz_labels)  
iv_labels_int = [class_to_int_mapping['IVE']] * len(iv_labels)  
lf_labels_int = [class_to_int_mapping['LeSserafim']] * len(lf_labels)  
nj_labels_int = [class_to_int_mapping['NewJeans']] * len(nj_labels)
```

✓ 0.0s

```
all_labels = np.array(ap_labels_int + bp_labels_int + cd_labels_int + iz_labels_int + iv_labels_int + lf_labels_int + nj_labels_int)  
all_labels = np.array(all_labels)
```

✓ 0.0s

```
(array([0, 0, ..., 6, 6, 6]), 90100, (90100,))
```

## 전 처리 과정

- Face extractor 페이지 활용을 통한 사진 얼굴 추출
- **Resize & gray-scale**
- **각 이미지 별 Label 설정 및 저장**
- **이미지 회전, 반전 등을 통한 100배 증폭 (901장 => 90,100장)**
- **Numpy 배열로 변환하여 저장**

# 4. Face Idol

## 학습 진행

```
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

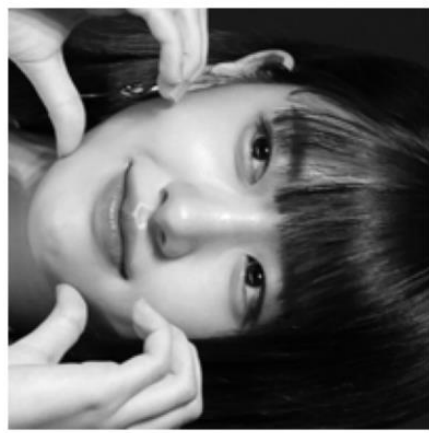
```
def predict_image(model, image_path, target_size=(200, 200)):
    try:
        img = Image.open(image_path)
        img = img.convert('L').resize(target_size)
        img_array = np.array(img)
        img_array = img_array.reshape((1, -1))
        prediction = model.predict(img_array)
        predicted_label_index = np.argmax(prediction)
        return predicted_label_index
    except Exception as e:
        print(f"Failed to predict image: {str(e)}")
        return None
```

```
def label_group(predicted_label):
    if predicted_label == 0:
        return('에스파')
    elif predicted_label == 1:
        return('블랙핑크')
    elif predicted_label == 2:
        return('아이들')
    elif predicted_label == 3:
        return('있지')
    elif predicted_label == 4:
        return('아이브')
    elif predicted_label == 5:
        return('르세라핌')
```

```
1/1 [=====] - 0s
예상 아이돌 그룹: 아이들
```

```
1/1 [=====] - 0s 469ms/step
```

예상 아이돌 그룹: 아이브



## 학습 과정

- DNN 학습
- Flatten 및 Dense 레이어 추가 후 compile
- Train, Validation 데이터 분리 학습 진행
- Test 데이터를 통해 확인
- 함수 생성 및 새로운 이미지 데이터로 확인 및 라벨 수정
- 모델 저장

```
model.fit(train_x, train_y, epochs=10, batch_size=100, validation_split=0.2)
```

✓ 3m 47.3s

```
Epoch 1/10
577/577 [=====] - 24s 38ms/step - loss: 1.9899 - accuracy: 0.2129 - val_loss: 1.8846 - val_accuracy: 0.2279
Epoch 2/10
577/577 [=====] - 19s 32ms/step - loss: 1.8139 - accuracy: 0.2710 - val_loss: 1.6970 - val_accuracy: 0.3356
Epoch 3/10
577/577 [=====] - 21s 36ms/step - loss: 1.5785 - accuracy: 0.3829 - val_loss: 1.3782 - val_accuracy: 0.4821
Epoch 4/10
577/577 [=====] - 20s 35ms/step - loss: 1.0717 - accuracy: 0.5961 - val_loss: 0.7386 - val_accuracy: 0.7219
Epoch 5/10
577/577 [=====] - 22s 39ms/step - loss: 0.5340 - accuracy: 0.8103 - val_loss: 0.3867 - val_accuracy: 0.8579
Epoch 6/10
577/577 [=====] - 23s 40ms/step - loss: 0.2100 - accuracy: 0.9367 - val_loss: 0.1018 - val_accuracy: 0.9752
Epoch 7/10
577/577 [=====] - 21s 37ms/step - loss: 0.1255 - accuracy: 0.9616 - val_loss: 0.0457 - val_accuracy: 0.9908
Epoch 8/10
577/577 [=====] - 21s 37ms/step - loss: 0.0947 - accuracy: 0.9705 - val_loss: 0.0092 - val_accuracy: 0.9994
Epoch 9/10
577/577 [=====] - 20s 34ms/step - loss: 0.1730 - accuracy: 0.9515 - val_loss: 0.0174 - val_accuracy: 0.9979
Epoch 10/10
577/577 [=====] - 16s 28ms/step - loss: 0.0144 - accuracy: 0.9976 - val_loss: 0.0749 - val_accuracy: 0.9729
```

<keras.callbacks.History at 0x2d88c45a790>





**THANK  
YOU**