

사진 속 표정 인식

1조

목차

- ▶ 주제 변경 이유
- ▶ 구현 방식
- ▶ 만든 모델 코드 설명

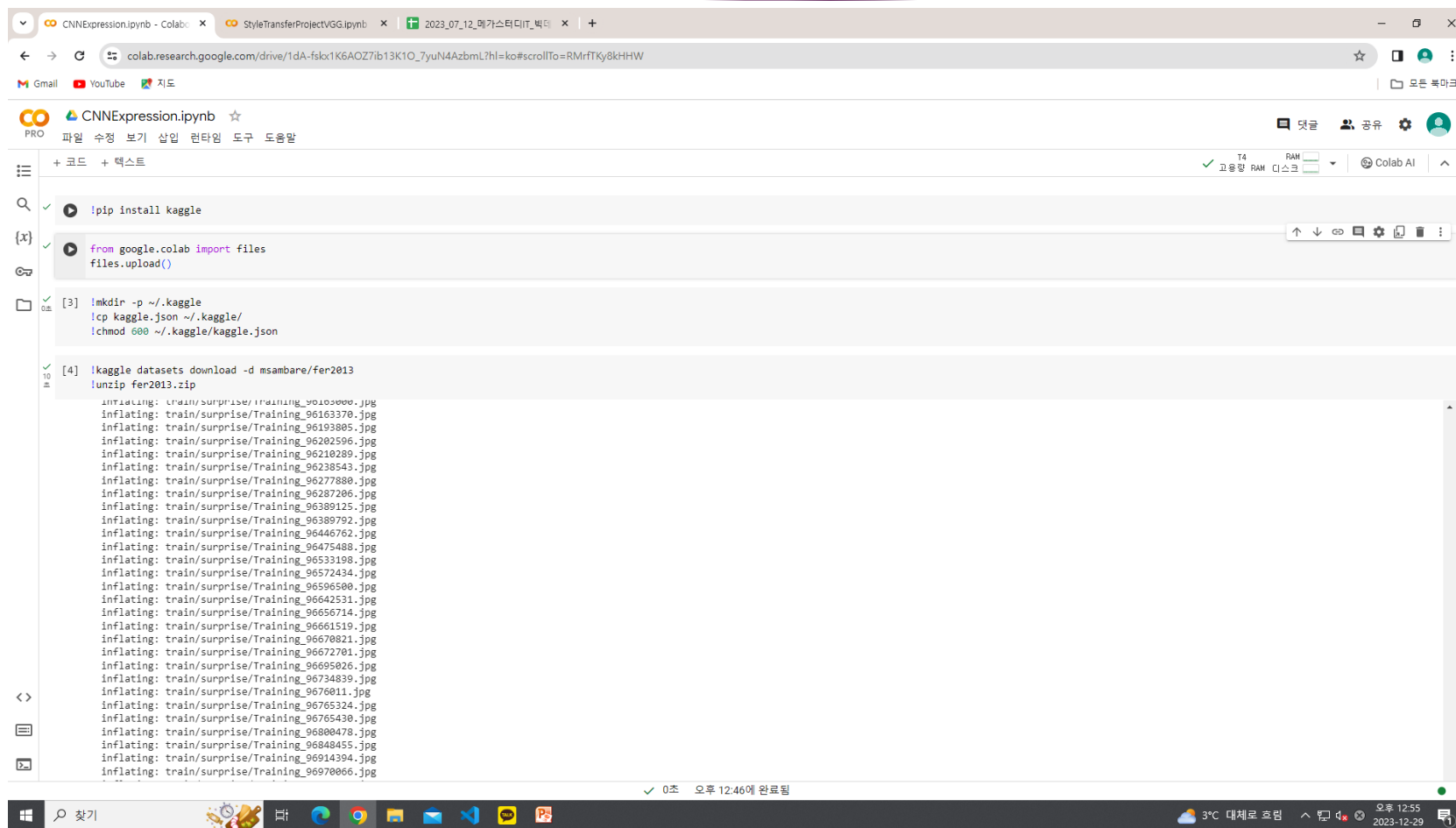
주제 변경 이유

- ▶ 팀원이 줄면서 원래 하려던 프로젝트 진행 불가
 - ▶ 나머지 둘이서 진행하기엔 이전 프로젝트의 복잡도가 너무 높음

구현 방식

- ▶ CNN을 이용한 모델 하나는 구축 완료
- ▶ 가능하면 더 다양한 방식을 이용해 표정 인식 모델 구현 계획

만든 모델 코드 설명



```
!pip install kaggle

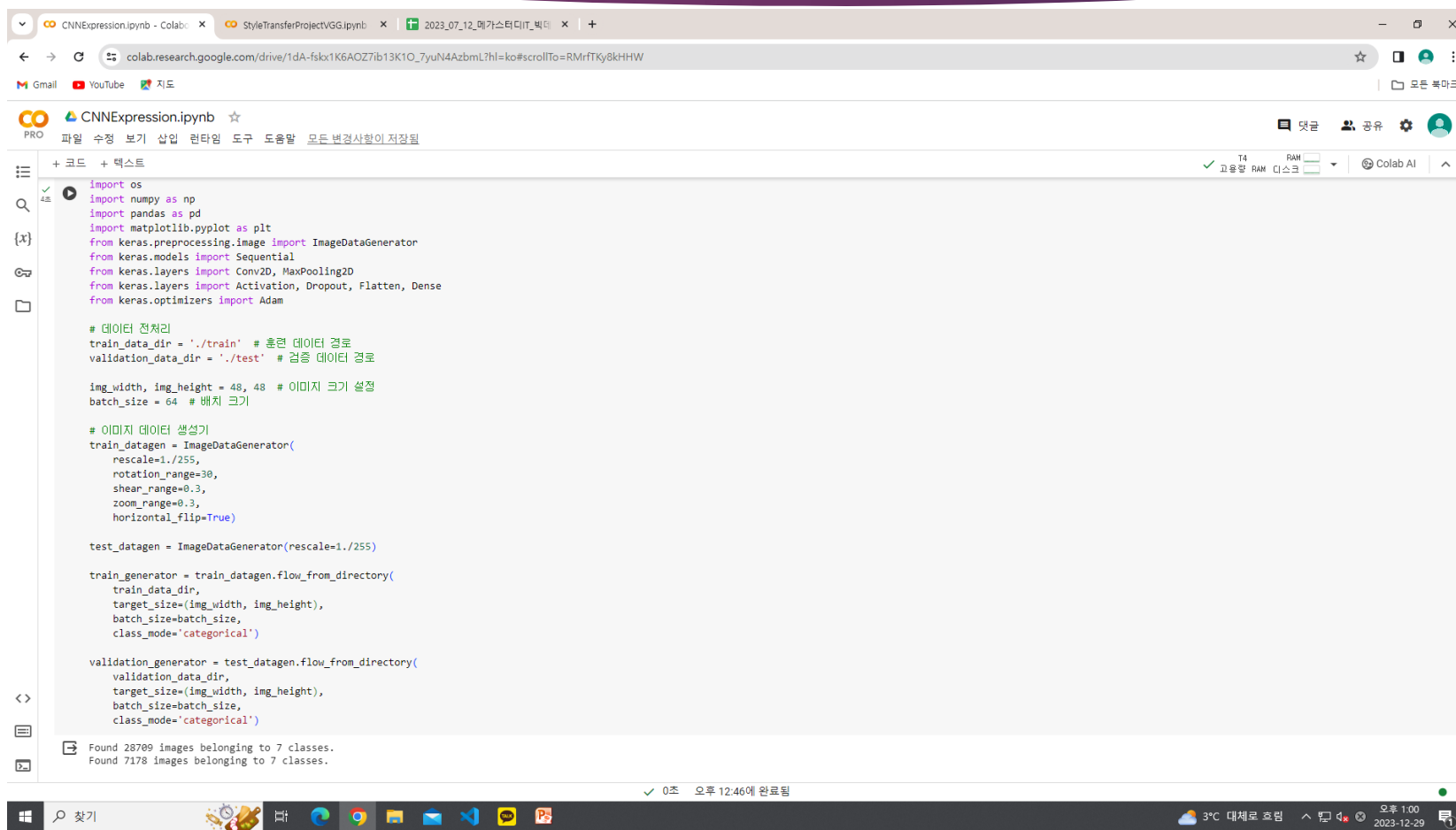
from google.colab import files
files.upload()

[3] !mkdir -p ~/.kaggle
     !cp kaggle.json ~/.kaggle/
     !chmod 600 ~/.kaggle/kaggle.json

[4] !kaggle datasets download -d msmbare/fer2013
     !unzip fer2013.zip

Inflating: train/surprise/training_90103000.jpg
Inflating: train/surprise/training_96163370.jpg
Inflating: train/surprise/training_96193805.jpg
Inflating: train/surprise/training_96202596.jpg
Inflating: train/surprise/training_96210289.jpg
Inflating: train/surprise/training_96238543.jpg
Inflating: train/surprise/training_96277880.jpg
Inflating: train/surprise/training_96287206.jpg
Inflating: train/surprise/training_96380125.jpg
Inflating: train/surprise/training_96389792.jpg
Inflating: train/surprise/training_96446762.jpg
Inflating: train/surprise/training_96475488.jpg
Inflating: train/surprise/training_96533198.jpg
Inflating: train/surprise/training_96572434.jpg
Inflating: train/surprise/training_96596500.jpg
Inflating: train/surprise/training_96642531.jpg
Inflating: train/surprise/training_96656714.jpg
Inflating: train/surprise/training_96661519.jpg
Inflating: train/surprise/training_96670821.jpg
Inflating: train/surprise/training_96672701.jpg
Inflating: train/surprise/training_96695026.jpg
Inflating: train/surprise/training_96734839.jpg
Inflating: train/surprise/training_96760111.jpg
Inflating: train/surprise/training_96765324.jpg
Inflating: train/surprise/training_96765430.jpg
Inflating: train/surprise/training_96800478.jpg
Inflating: train/surprise/training_96840455.jpg
Inflating: train/surprise/training_96914394.jpg
Inflating: train/surprise/training_96970066.jpg
```

만든 모델 코드 설명



```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam

# 데이터 전처리
train_data_dir = './train' # 훈련 데이터 경로
validation_data_dir = './test' # 검증 데이터 경로

img_width, img_height = 48, 48 # 이미지 크기 설정
batch_size = 64 # 배치 크기

# 이미지 데이터 생성기
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

Found 28789 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.

만든 모델 코드 설명

```

[6] model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(img_width, img_height, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7)) # 7개의 표정 클래스
    model.add(Activation('softmax'))

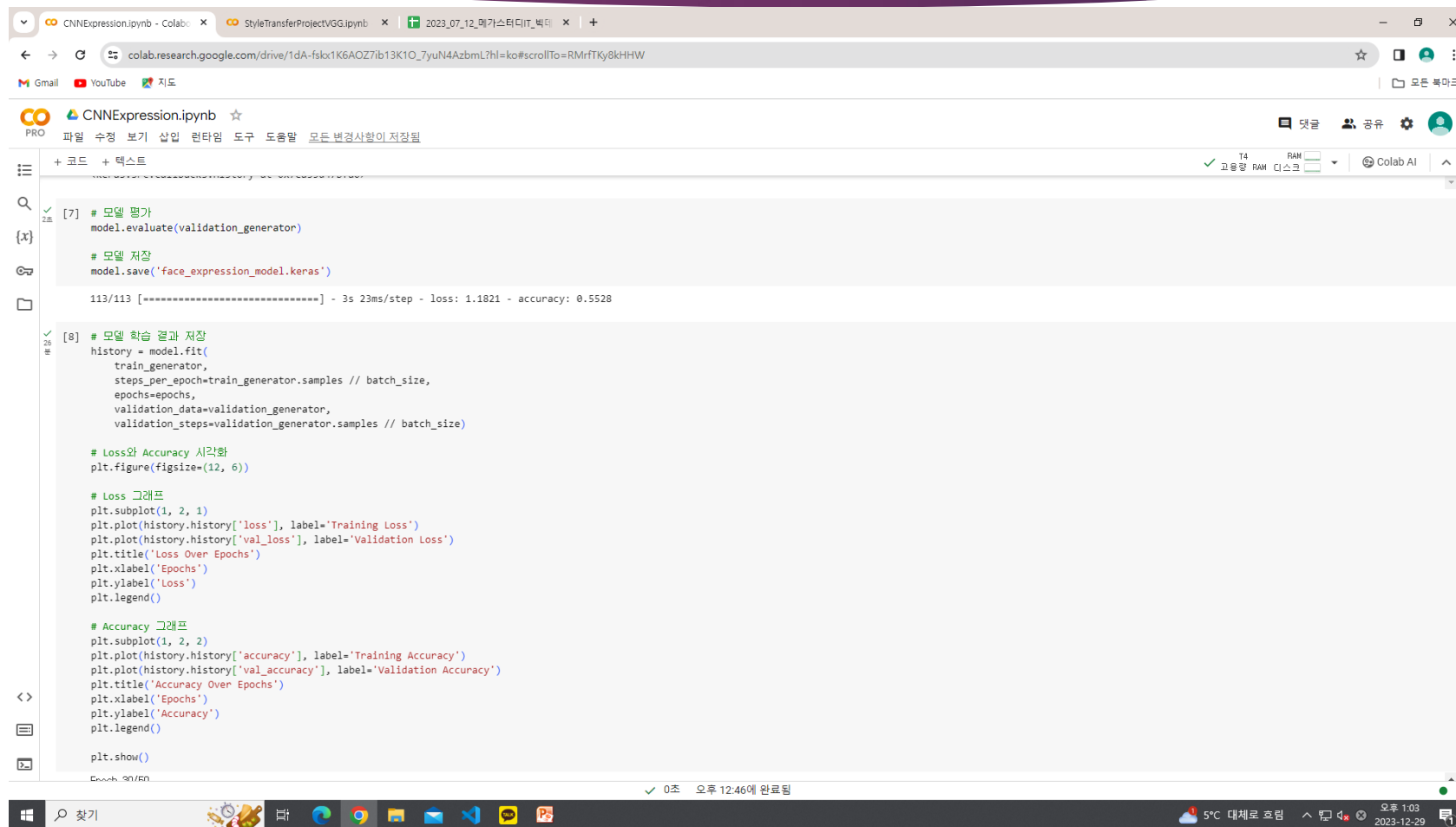
    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(lr=0.0001),
                  metrics=['accuracy'])

# 모델 학습
epochs = 50
model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)

448/448 [=====] - 32s 71ms/step - loss: 1.3674 - accuracy: 0.4790 - val_loss: 1.2350 - val_accuracy: 0.5339
Epoch 23/50
448/448 [=====] - 32s 72ms/step - loss: 1.3618 - accuracy: 0.4818 - val_loss: 1.2264 - val_accuracy: 0.5225
Epoch 24/50
448/448 [=====] - 32s 72ms/step - loss: 1.3614 - accuracy: 0.4802 - val_loss: 1.2422 - val_accuracy: 0.5294
Epoch 25/50
448/448 [=====] - 32s 72ms/step - loss: 1.3575 - accuracy: 0.4824 - val_loss: 1.2330 - val_accuracy: 0.5338
Epoch 26/50
448/448 [=====] - 32s 71ms/step - loss: 1.3513 - accuracy: 0.4850 - val_loss: 1.2198 - val_accuracy: 0.5377

```

만든 모델 코드 설명



```
[7] # 모델 평가
model.evaluate(validation_generator)

# 모델 저장
model.save('face_expression_model.keras')

113/113 [=====] - 3s 23ms/step - loss: 1.1821 - accuracy: 0.5528

[8] # 모델 학습 결과 저장
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)

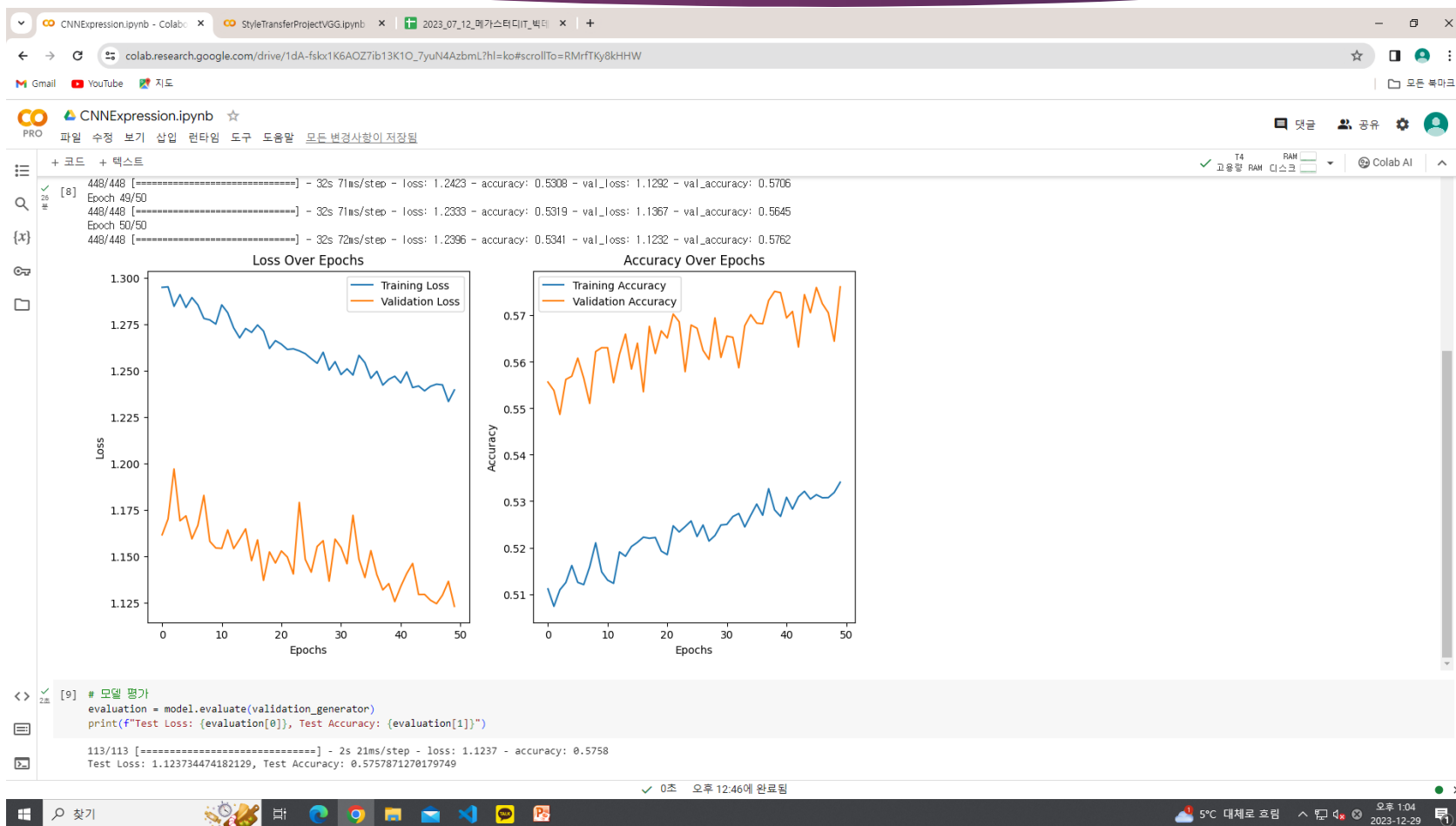
# Loss와 Accuracy 시각화
plt.figure(figsize=(12, 6))

# Loss 그래프
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

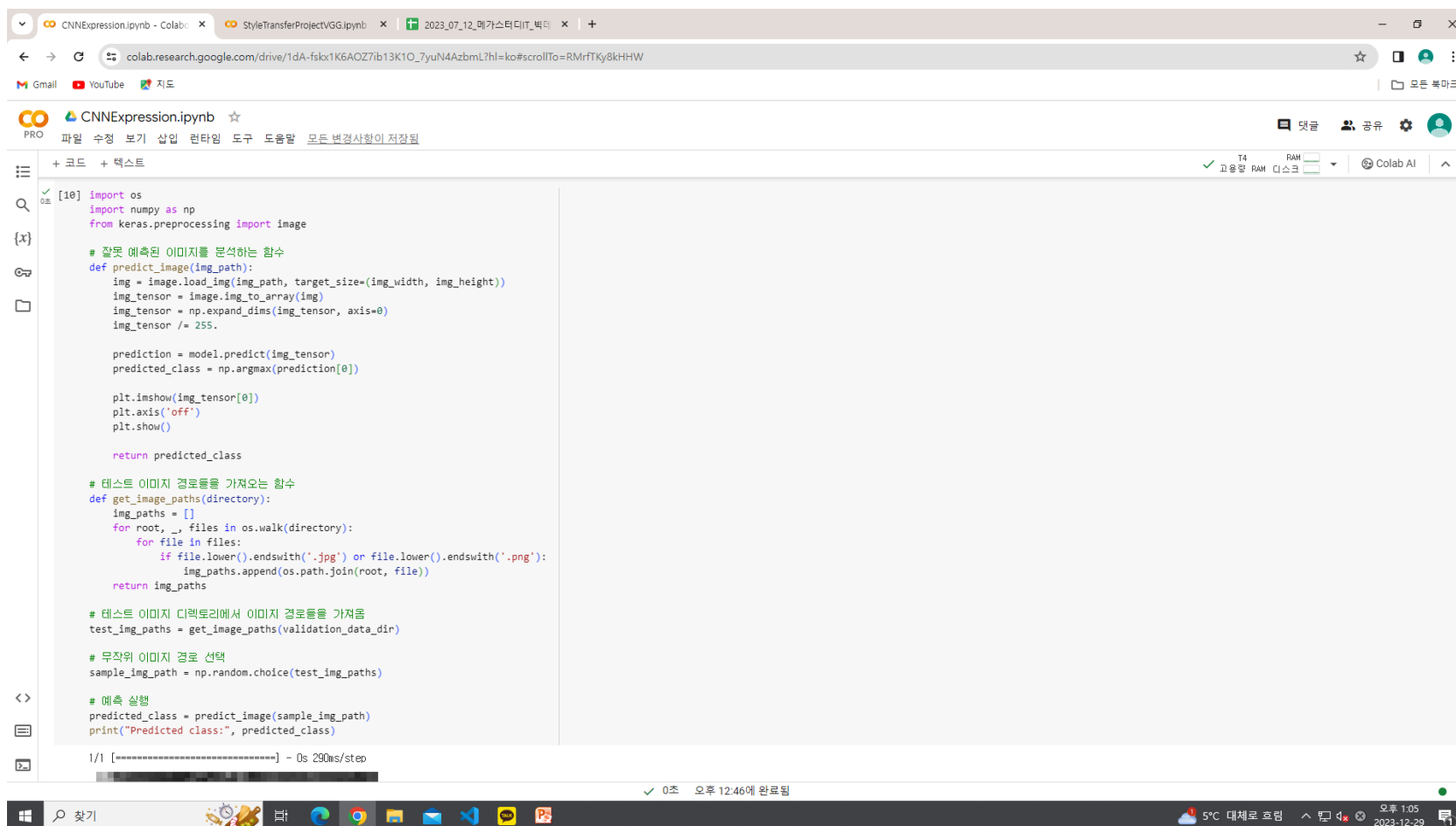
# Accuracy 그래프
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```


만든 모델 코드 설명



만든 모델 코드 설명



```
[10] import os
import numpy as np
from keras.preprocessing import image

# 잘못 예측된 이미지를 분석하는 함수
def predict_image(img_path):
    img = image.load_img(img_path, target_size=(img_width, img_height))
    img_tensor = image.img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    prediction = model.predict(img_tensor)
    predicted_class = np.argmax(prediction[0])

    plt.imshow(img_tensor[0])
    plt.axis('off')
    plt.show()

    return predicted_class

# 테스트 이미지 경로를 가져오는 함수
def get_image_paths(directory):
    img_paths = []
    for root, _, files in os.walk(directory):
        for file in files:
            if file.lower().endswith('.jpg') or file.lower().endswith('.png'):
                img_paths.append(os.path.join(root, file))
    return img_paths

# 테스트 이미지 디렉토리에서 이미지 경로를 가져옴
test_img_paths = get_image_paths(validation_data_dir)

# 무작위 이미지 경로 선택
sample_img_path = np.random.choice(test_img_paths)

# 예측 실행
predicted_class = predict_image(sample_img_path)
print("Predicted class:", predicted_class)

1/1 [=====] - 0s 290ms/step
```

만든 모델 코드 설명

Colab interface showing a Jupyter Notebook titled "CNNExpression.ipynb". The notebook displays a grayscale image of a person's face and the output "Predicted class: 4". The interface includes a file explorer on the left, a code editor with a single cell, and a status bar at the bottom indicating "0초" (0 seconds) and "오후 12:46에 완료됨" (Completed at 12:46 PM).

Browser tabs: CNNExpression.ipynb - Colab, StyleTransferProjectVGG.ipynb, 2023_07_12_메가스터디IT_빅테

Address bar: colab.research.google.com/drive/1dA-fsloX1K6AOZ7ib13K1O_7yuN4AzbmL?hl=ko#scrollTo=RMrfTKy8KHHW

Navigation: Gmail, YouTube, 지도

Colab interface: CNNExpression.ipynb, 파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

Code cell output: [10] 1/1 [=====] - 0s 290ms/step

Image: A grayscale image of a person's face.

Predicted class: 4

Message: [] 코딩을 시작하거나 AI로 코드를 생성하세요.

Status bar: 0초 오후 12:46에 완료됨

Windows taskbar: 5°C 대체로 흐림, 오후 1:07, 2023-12-29

중간발표 - 얼굴 표정 인식

1조 - 김현범, 좌준호

데이터 수집 및 전처리

[illegible]

```
data['Usage'].value_counts()
```

Training	28709
PublicTest	3589
PrivateTest	3589



훈련데이터:28709개
검증데이터:3589개
테스트데이터:3589개

데이터 수집 및 전처리

```
def prepare_data(data):  
    """ Prepare data for modeling  
    input: data frame with labels und pixel data  
    output: image and label array """  
  
    image_array = np.zeros(shape=(len(data), 48, 48))  
    # 0으로만 채워진 3차원 array를 생성합니다. 48행 48열, data개수만큼  
    image_label = np.array(list(map(int, data['emotion'])))  
  
    for i, row in enumerate(data.index):  
        image = np.fromstring(data.loc[row, 'pixels'], dtype=int, sep=' ')  
        # the fromstring() function is used to create a new 1D (one-dimensional) array from a string that contains data,  
        image = np.reshape(image, (48, 48))  
        # reshape함수는 현재의 배열의 차원(1차원, 2차원, 3차원)을 변경하여 행렬을 반환하거나 하는 경우에 많이 이용되는 함수입니다.  
        # image 1차원 배열을 48행 48열 행렬로 변환합니다.  
        image_array[i] = image  
  
    return image_array, image_label
```

```
train_image_array, train_image_label = prepare_data(data[data[' Usage']=='Training'])  
val_image_array, val_image_label = prepare_data(data[data[' Usage']=='PrivateTest'])  
test_image_array, test_image_label = prepare_data(data[data[' Usage']=='PublicTest'])
```

1. 문자열 형태의 픽셀 데이터
-> 정수배열로 변환
2. 1차원 이미지 배열을
2차원배열로 구성
3. 이미지배열과 레이블배열
생성 후 저장

데이터 수집 및 전처리

```
train_images = train_image_array.reshape((train_image_array.shape[0], 48, 48, 1))
train_images = train_images.astype('float32')/255
val_images = val_image_array.reshape((val_image_array.shape[0], 48, 48, 1))
val_images = val_images.astype('float32')/255
test_images = test_image_array.reshape((test_image_array.shape[0], 48, 48, 1))
test_images = test_images.astype('float32')/255
```

1. 원래 배열의 첫번째 차원크기를 유지

➔ 각 이미지를 48x48 크기의 단일 채널 이미지로 변환

➔ CNN이 요구하는 입력형식에 맞추는 과정

2. 이미지데이터를 float32타입으로 변환 후 픽셀값을 255로 나누어 정규화

-> 데이터를 0과 1사의 값으로 스케일링하여 모델이 더 잘 학습되도록 도와줌

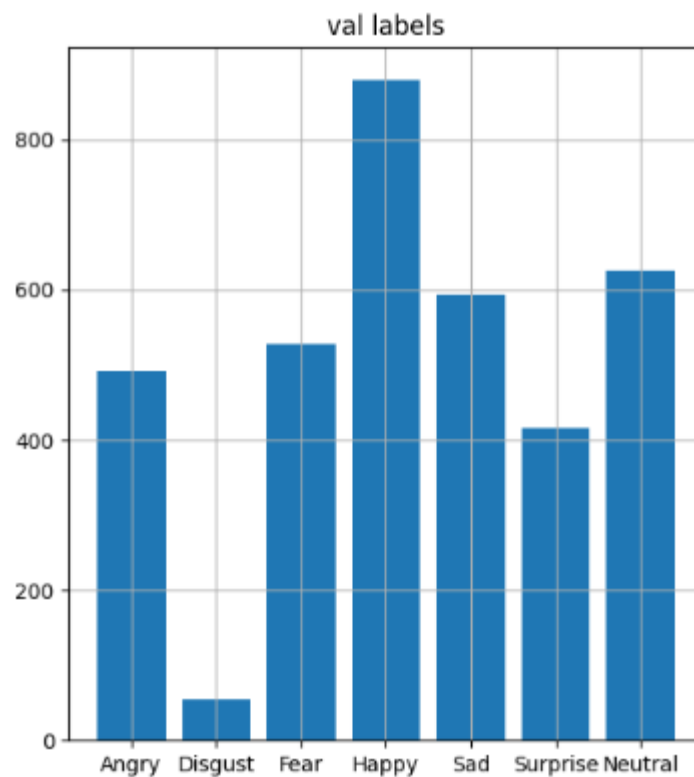
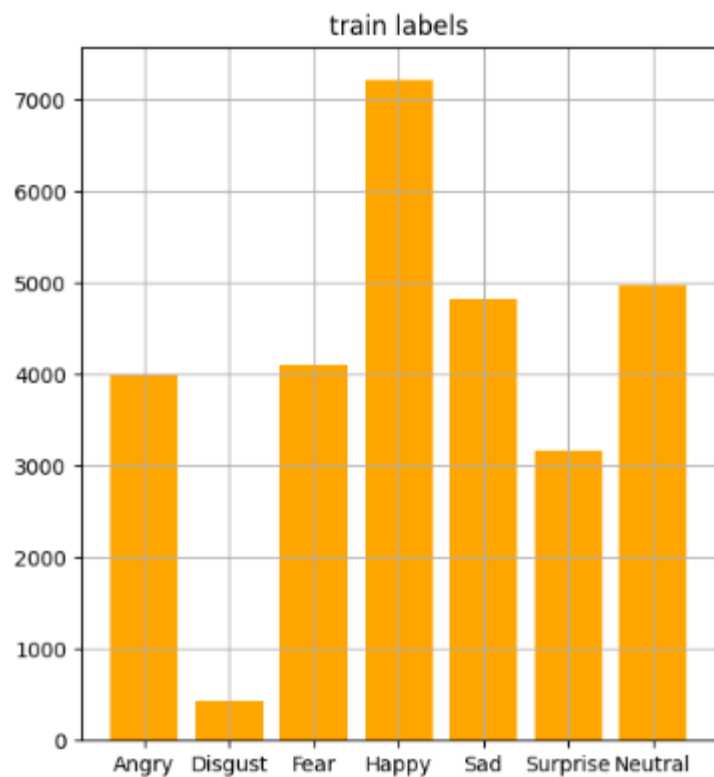
데이터 수집 및 전처리

```
train_labels = to_categorical(train_image_label)
val_labels = to_categorical(val_image_label)
test_labels = to_categorical(test_image_label)
```

1. 정수로 된 레이블 배열을 원-핫 인코딩된 형태로 변환
-> 각 레이블은 이진 배열로 변환되며 해당 레이블에 해당하는 위치는 1이고 나머지는 0이된다.

데이터 수집 및 전처리

```
plot_compare_distributions(train_labels, val_labels, title='train labels', title2='val labels')
```



1. 'Happy' 감정의 레이블이 가장 많이 등장
2. 'Disgust'는 다른 감정에 비해 상대적으로 적은 빈도로 나타남.
3. 'Angry', 'Fear', 'Sad', 'Surprise', 'Neutral' 감정들은 비교적 균등하게 분포하나 'Happy' 다음으로 'Sad'와 'Neutral' 레이블의 빈도가 높게 나타나는 경향을 보임
4. -> 검증데이터도 같음

데이터 수집 및 전처리

```
class_weight = dict(zip(range(0, 7), (((data[data[' Usage']=='Training']['emotion'].value_counts()).sort_index())/len(data[data[' Usage']=='Training']['emotion'])).tolist()))
```

1. 'Usage' 열이 'Training'인 행을 필터링하여 훈련 데이터를 선택하고, 'emotion' 열에 대해 각 감정 레이블의 빈도를 계산
2. 레이블의 빈도를 인덱스(일반적으로 감정 레이블을 나타내는 정수)에 따라 정렬
3. 각 레이블 빈도를 훈련 데이터셋의 전체 길이로 나누어 각 클래스의 상대적인 비율을 계산
4. 계산된 비율을 리스트로 변환
5. 0부터 6까지의 숫자(7개의 감정 클래스를 나타냄)와 위에서 계산한 비율 리스트를 짝지어 사전 생성
6. 'class-weight' 는 각 클래스 인덱스를 키로, 해당 클래스의 상대적인 비율을 값으로 가지게 됨

->이 사전은 머신 러닝 모델의 훈련 과정에서 클래스 불균형을 조정하는 데 사용
-> 더 적은 데이터 포인트를 가진 클래스에 더 큰 손실 가중치를 부여하여, 그 클래스의 중요성을 인공적으로 높이는 데 사용

class_weight

```
{0: 0.1391549688251071,  
1: 0.01518687519593159,  
2: 0.14270786164617366,  
3: 0.2513149186666202,  
4: 0.16823992476226968,  
5: 0.11045316799609878,  
6: 0.17294228290779895}
```

CNN모델 구현하기

```
model = models.Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(7, activation='softmax'))

model.compile(optimizer=Adam(lr=1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 46, 46, 64)	640
max_pooling2d (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_1 (Conv2D)	(None, 21, 21, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 10, 10, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 2, 2, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 1, 1, 256)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 256)	65792
dense_1 (Dense)	(None, 128)	32896
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 7)	903

```
=====  
Total params: 506183 (1.93 MB)  
Trainable params: 506183 (1.93 MB)  
Non-trainable params: 0 (0.00 Byte)
```

CNN모델 구현하기

```
history = model.fit(train_images, train_labels,  
                    validation_data=(val_images, val_labels),  
                    class_weight = class_weight,  
                    epochs=200,  
                    batch_size=64)
```



```
test_loss, test_acc = model.evaluate(test_images, test_labels)  
print('test caccuracy:', test_acc)
```

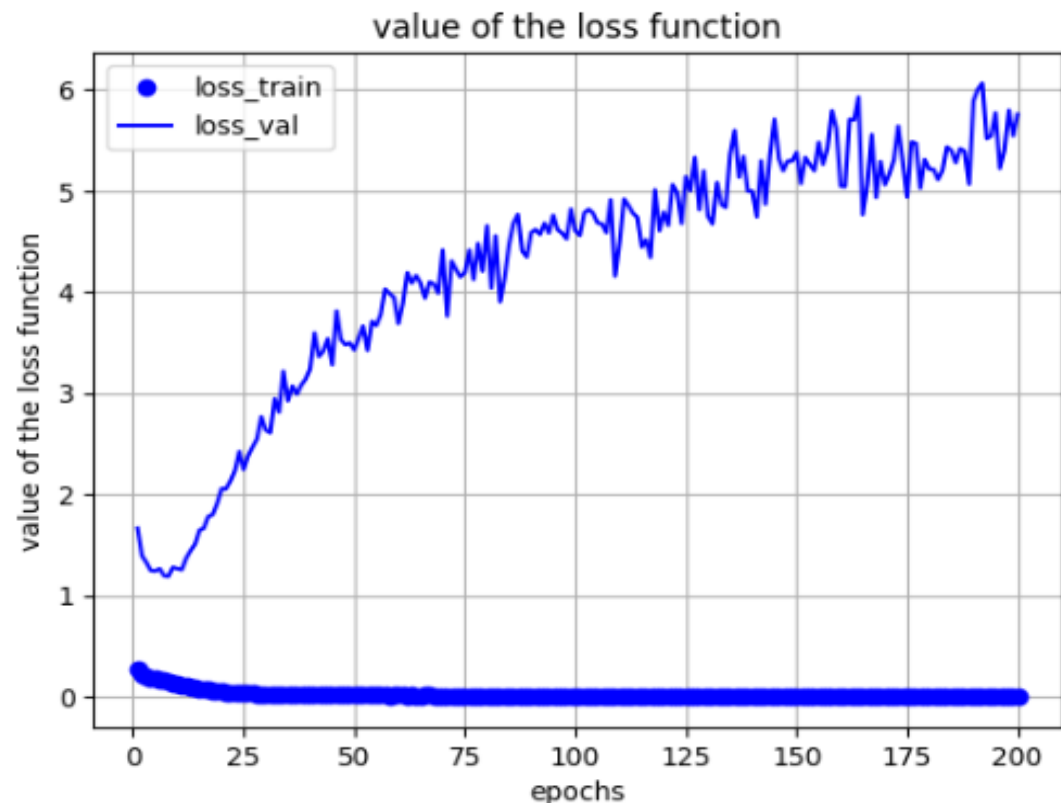
```
113/113 [=====] - 1s 4ms/step - loss: 5.7935 - accuracy: 0.5600  
test caccuracy: 0.5600445866584778
```

손실값: 5.7
정확도:0.56

Convergence 분석

```
loss = history.history['loss']
loss_val = history.history['val_loss']
epochs = range(1, len(loss)+1)
plt.plot(epochs, loss, 'bo', label='loss_train')
plt.plot(epochs, loss_val, 'b', label='loss_val')
plt.title('value of the loss function')
plt.xlabel('epochs')
plt.ylabel('value of the loss function')
plt.legend()
plt.grid()
plt.show()
```

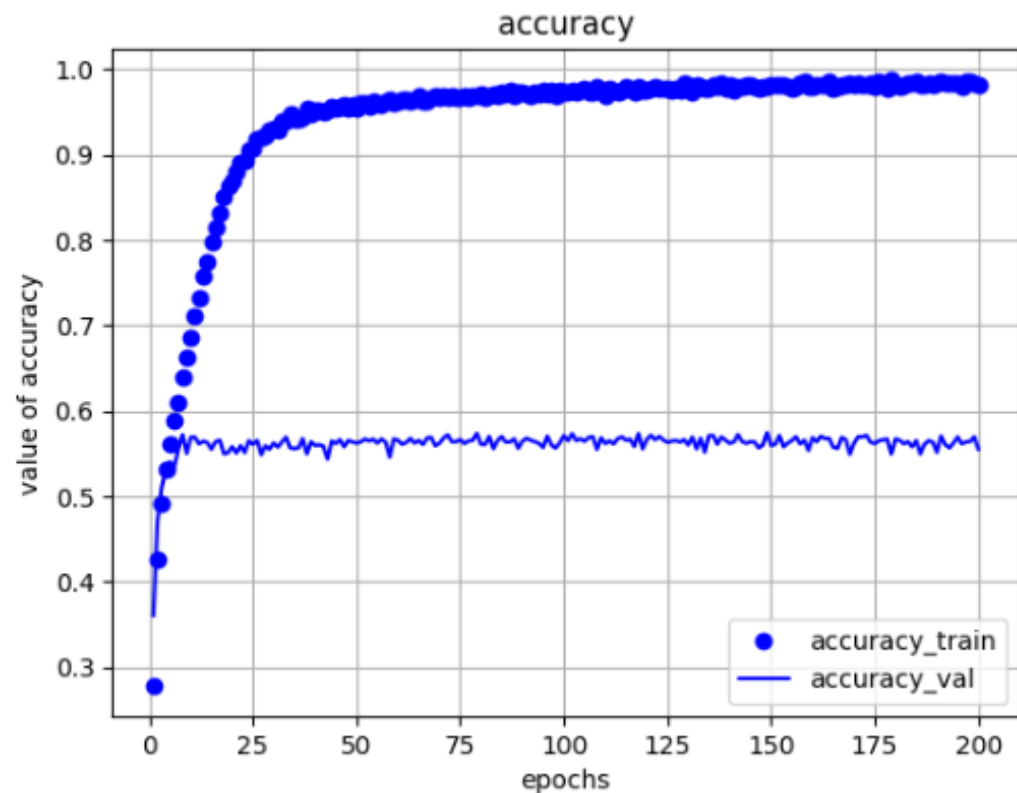
1. 초기 에포크에서는 두 손실 값 모두 빠르게 감소
-> 모델 초기에 많이 배우고 있음을 나타냄
 2. 훈련 손실은 지속적으로 감소
-> 모델 훈련이 잘 맞춰지고 있음을 의미
 3. 검증 손실은 에포크가 증가함에 따라 줄어들다가 약간 증가하는 경향을 보임.
-> 어느정도 에포크 이후에는 검증 데이터셋에 대해 더 이상 큰 개선을 보이지 않음 -> 과적합 신호일수 있음
- 파란색점: 훈련 손실값 / 파란색 선 : 검증 손실



Convergence 분석

```
acc = history.history['accuracy']
acc_val = history.history['val_accuracy']
epochs = range(1, len(loss)+1)
plt.plot(epochs, acc, 'bo', label='accuracy_train')
plt.plot(epochs, acc_val, 'b', label='accuracy_val')
plt.title('accuracy')
plt.xlabel('epochs')
plt.ylabel('value of accuracy')
plt.legend()
plt.grid()
plt.show()
```

1. 훈련 정확도 매우 빠르게 상승, 그 이후 변화 없음
➔ 모델이 훈련데이터에 빠르게 학습 후 개선의 여지없음
 2. 검증데이터도 비슷한 패턴을 보이며 훈련 정확도와 일치 -> 잘 일반화되고 있음
 3. 두 정확도 사이에 큰 간격이 없음.
-> 과적합에 문제되지 않음
- 파란색점: 훈련 정확도의 에포크 값 / 파란색 선 : 검증 정확도

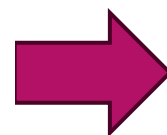
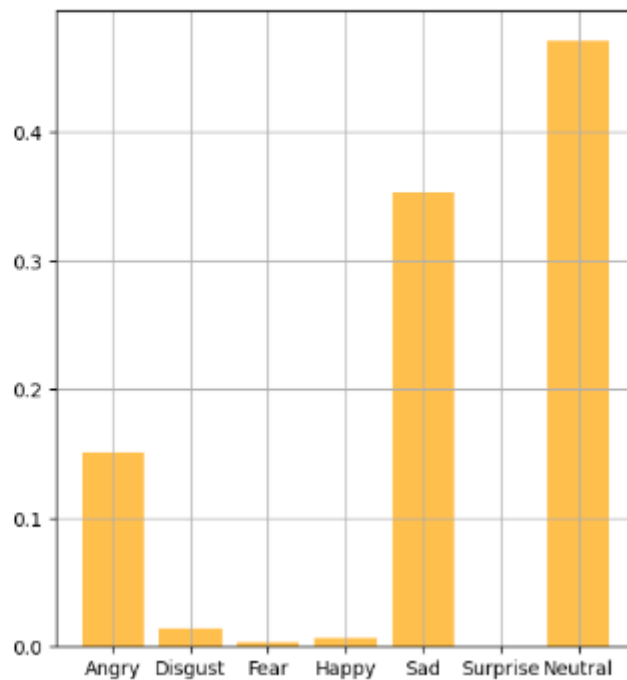
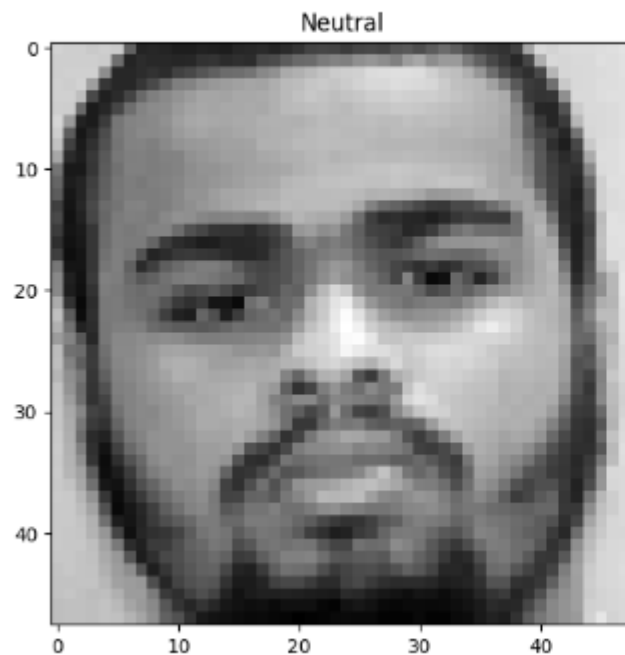


최종발표 - 얼굴 표정 인식

1조 - 김현범, 좌준호

결과 분석

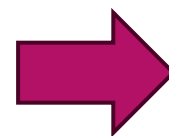
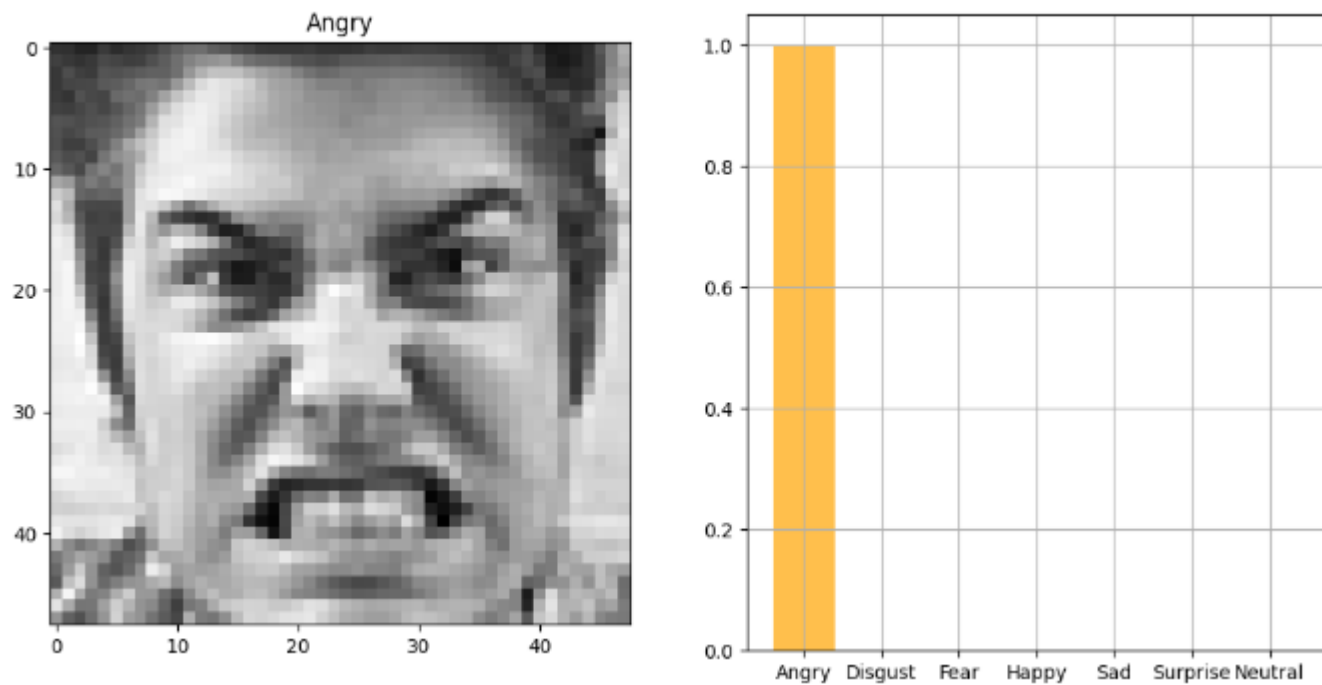
```
[30] plot_image_and_emotion(test_image_array, test_image_label, pred_test_labels, 106)
```



예측: Neutral
'sad'와 'angry'도 상대적으로 높은 확률 값을 가지고 있다.

결과 분석

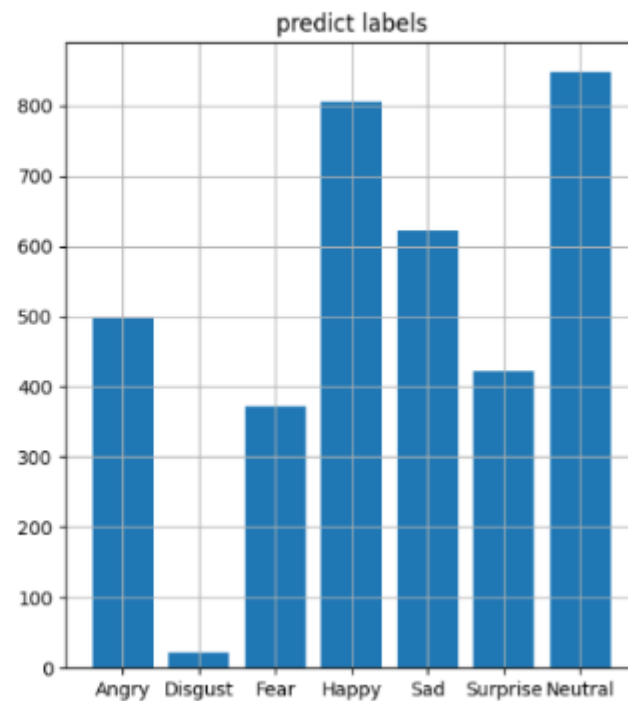
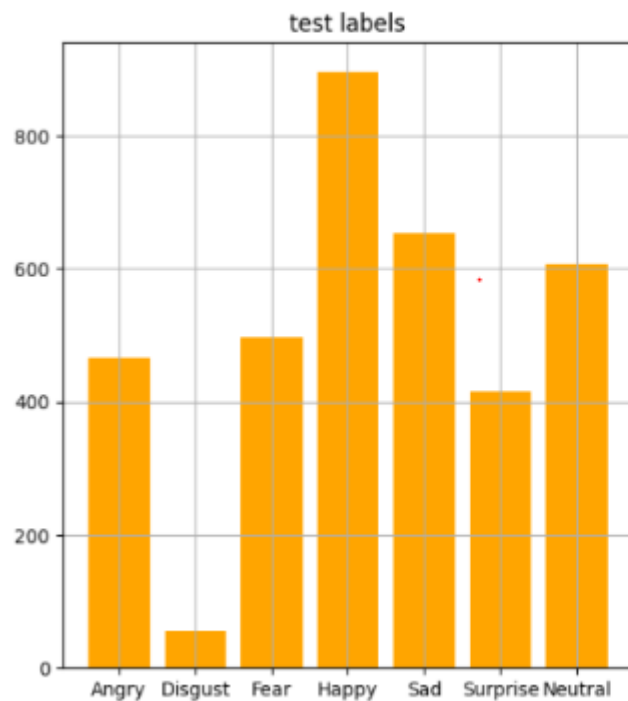
```
plot_image_and_emotion(test_image_array, test_image_label, pred_test_labels, 40)
```



예측:Angry
모델이 이미지 속 표정을 'Angry'로
매우 확신하고 있음을 나타냄

결과 분석

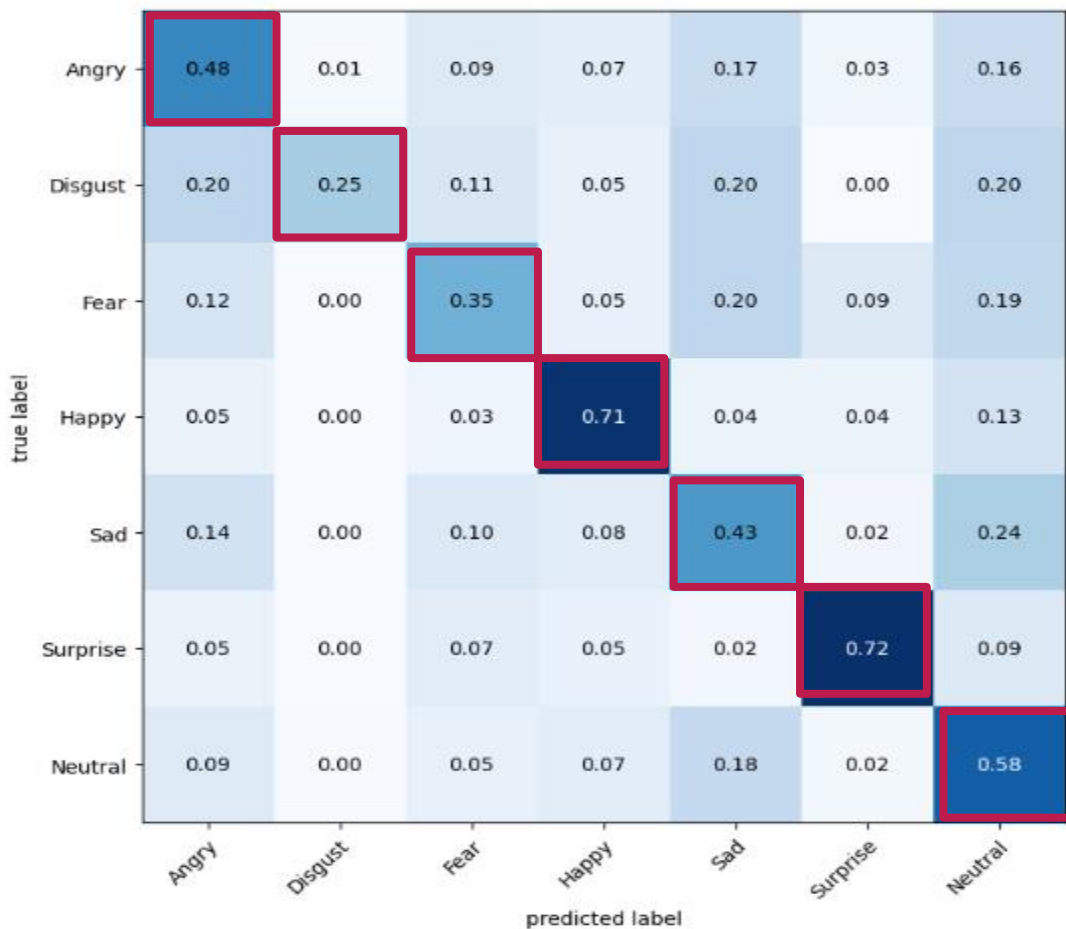
```
plot_compare_distributions(test_labels, pred_test_labels, title1='test labels', title2='predict labels')
```



왼쪽 그래프 - 테스트 레이블: 'Happy' 감정이 가장 많은 빈도로 나타나며, 'Disgust' 감정이 가장 적게 나타냄

오른쪽 그래프 - 예측 레이블: 'Neutral'과 'Happy'를 가장 많이 예측하는 경향이 있는 것으로 보이며 'Fear'와 'Sad' 감정에 대한 예측은 상대적으로 적음

혼동 행렬



• 'Angry'의 실제 레이블에 대해 모델은 48%의 경우에 올바르게 'Angry'라고 예측

• 'Happy'의 경우에는 71%의 경우에 모델이 올바르게 'Happy'라고 예측 'Surprise'와 'Neutral' 레이블도 각각 72%와 58%의 비율로 높은 정확도를 보여줌

• 'Disgust' 레이블에 대해서는 모델이 25%의 경우에만 올바르게 예측했으며, 다른 감정과 혼동할 가능성이 높음을 알 수 있음

다른 데이터 수집 후 모델 적용

```
▶ model.save('face_recognition_model.h5') # 모델을 HDF5 파일로 저장  
📄 /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: I  
    saving_api.save_model(  
    
```



모델 저장

```
[36] from keras.models import load_model  
  
# 모델을 새로운 객체로 불러옵니다  
loaded_model = load_model('face_recognition_model.h5')
```

```
[37] from keras.preprocessing import image  
import numpy as np  
  
def load_and_preprocess_image(image_path):  
    img = image.load_img(image_path, target_size=(48, 48), color_mode="grayscale")  
    img = image.img_to_array(img)  
    img = np.expand_dims(img, axis=0)  
    img /= 255 # Normalize to [0, 1]  
    return img  
  
# 예시 이미지 경로  
image_path = '/content/angry.PNG' # 경로에 이미지가 저장되어 있다고 가정합니다.  
  
# 이미지를 불러오고 전처리합니다  
test_image = load_and_preprocess_image(image_path)
```



객체 불러온 후 이미지 전처리

다른 데이터 수집 후 모델 적용



모델을 사용하여 예측 수행

```
predictions = loaded_model.predict(test_image)
predicted_class = np.argmax(predictions, axis=1)
predicted_emotion = emotions[predicted_class[0]]

print("Predicted Emotion:", predicted_emotion)
```

```
1/1 [=====] - 0s 301ms/step
Predicted Emotion: Angry
```



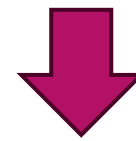
예측:Angry (O)

다른 데이터 수집 후 모델 적용



```
[C3] # 예시 이미지 경로  
image_path = "/content/disgust.PNG" # 경로에 이미지  
  
# 이미지를 불러오고 전처리합니다  
test_image = load_and_preprocess_image(image_path)  
  
# 모델을 사용하여 예측 수행  
predictions = loaded_model.predict(test_image)  
predicted_class = np.argmax(predictions, axis=1)  
predicted_emotion = emotions[predicted_class[0]]  
  
print("Predicted Emotion:", predicted_emotion)
```

```
1/1 [=====] - 0s 26ms/step  
Predicted Emotion: Sad
```



예측:Sad(틀림)

다른 데이터 수집 후 모델 적용



```
[40] # 예시 이미지 경로
      image_path = '/content/fear.PNG' # 경로에 이미지가 저장되어 있다고 가정합니다.

      # 이미지를 불러오고 전처리합니다
      test_image = load_and_preprocess_image(image_path)

      # 모델을 사용하여 예측 수행
      predictions = loaded_model.predict(test_image)
      predioted_class = np.argmax(predictions, axis=1)
      predioted_emotion = emotions[predioted_class[0]]

      print("Predioted Emotion:", predioted_emotion)

1/1 [=====] - 0s 26ms/step
Predioted Emotion: Fear
```



예측:Fear (O)

다른 데이터 수집 후 모델 적용



```
# 예시 이미지 경로
image_path = '/content/happy.PNG' # 경로에 이미지가 저장되어 있다고 가정합니다.

# 이미지를 불러오고 전처리합니다
test_image = load_and_preprocess_image(image_path)

# 모델을 사용하여 예측 수행
predictions = loaded_model.predict(test_image)
predioted_class = np.argmax(predictions, axis=1)
predioted_emotion = emotions[predioted_class[0]]

print("Predioted Emotion:", predioted_emotion)
```

1/1 [=====] - 0s 25ms/step
Predioted Emotion: Happy



예측:Happy (O)

다른 데이터 수집 후 모델 적용



```
# 예시 이미지 경로
image_path = '/content/neutral.PNG' # 경로에 이미지가 저장되어 있다고 가정합니다.

# 이미지를 불러오고 전처리합니다
test_image = load_and_preprocess_image(image_path)

# 모델을 사용하여 예측 수행
predictions = loaded_model.predict(test_image)
predioted_class = np.argmax(predictions, axis=1)
predioted_emotion = emotions[predioted_class[0]]

print("Predioted Emotion:", predioted_emotion)
```

1/1 [=====] - 0s 26ms/step
Predioted Emotion: Surprise



예측: Surprise(틀림)

다른 데이터 수집 후 모델 적용



```
[43] # 예시 이미지 경로
      image_path = '/content/sad.PNG' # 경로에 이미지가 저장되어 있다고 가정합니다.

      # 이미지를 불러오고 전처리합니다
      test_image = load_and_preprocess_image(image_path)

      # 모델을 사용하여 예측 수행
      predictions = loaded_model.predict(test_image)
      predicted_class = np.argmax(predictions, axis=1)
      predicted_emotion = emotions[predicted_class[0]]

      print("Predicted Emotion:", predicted_emotion)
```

```
1/1 [=====] - 0s 27ms/step
Predicted Emotion: Happy
```



예측:Happy(틀림)

다른 데이터 수집 후 모델 적용



```
# 예시 이미지 경로
image_path = '/content/surprise.PNG' # 경로에 이미지가 저장되어 있다고 가정합니다.

# 이미지를 불러오고 전처리합니다
test_image = load_and_preprocess_image(image_path)

# 모델을 사용하여 예측 수행
predictions = loaded_model.predict(test_image)
predioted_class = np.argmax(predictions, axis=1)
predioted_emotion = emotions[predioted_class[0]]

print("Predioted Emotion:", predioted_emotion)

1/1 [=====] - 0s 26ms/step
Predioted Emotion: Neutral
```



예측:Neutral(틀림)

결론

- 모델 학습 및 생성 완료
- 새로운 사진들에 대한 예측률이 많이 떨어지며 ;Disgust' 와 'Fear' 에 대해서는 예측률이 떨어짐
- 해당 레이블에 대한 더 많은 훈련 데이터를 수집하거나, 특징 추출 방식을 개선할 필요가 있음