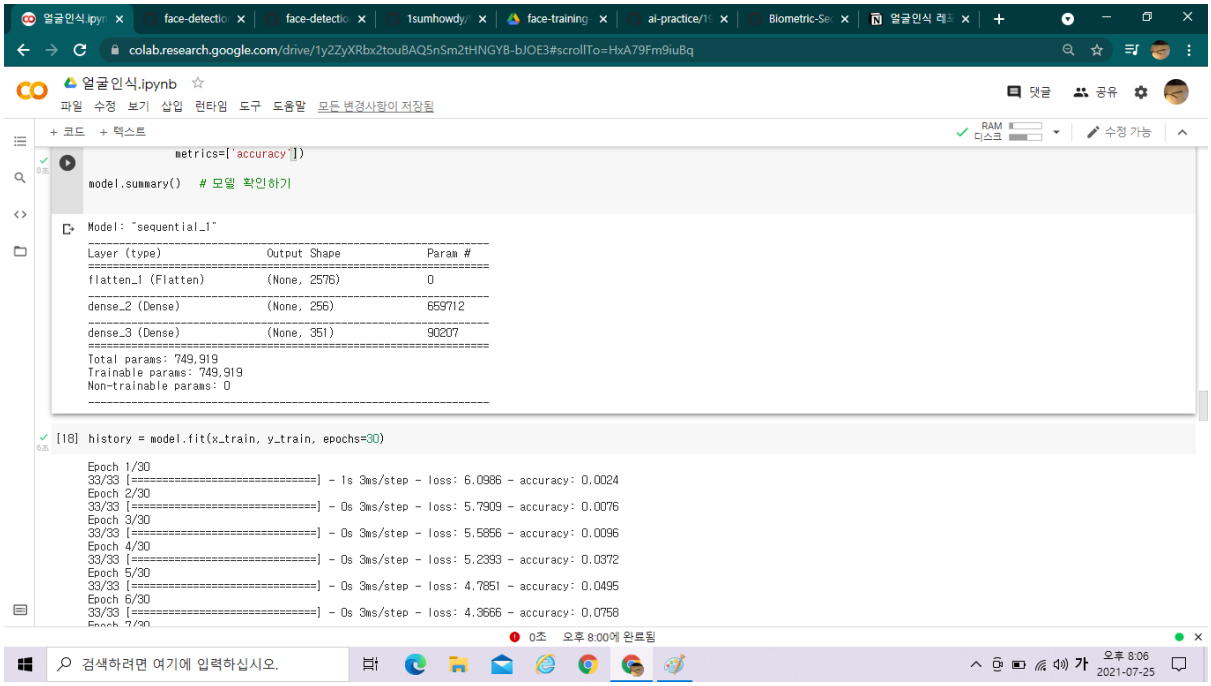
얼굴인식 레포트

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(56, 46)),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(351, activation='softmax')
])

# loss 함수로 sparse_categorical_crossentropy 사용
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()   # 모델 확인하기
```

layer는 많이 안 쌓고 기본적인 모델로 해보았다.



첫번째 모델 생성, 드라이브의 케라스 불러와서 실행하였다.

```
model.fit(x_train, y_train, epochs=30)
```

모델에 x_train, y_train을 학습시켰다. (epochs=30(30번)으로 설정함)

```
Epoch 1/30
33/33 [==============================] - 15s 3ms/step - loss: 6.0857 - accuracy: 0.0054
Epoch 2/30
33/33 [==============================] - 0s 3ms/step - loss: 5.7702 - accuracy: 0.0046
Epoch 3/30
33/33 [==============================] - 0s 3ms/step - loss: 5.5319 - accuracy: 0.0109
Epoch 4/30
33/33 [==============================] - 0s 3ms/step - loss: 5.1314 - accuracy: 0.0247
Epoch 5/30
33/33 [==============================] - 0s 3ms/step - loss: 4.5952 - accuracy: 0.0622
Epoch 6/30
33/33 [==============================] - 0s 3ms/step - loss: 4.1834 - accuracy: 0.1243
Epoch 7/30
33/33 [==============================] - 0s 3ms/step - loss: 3.7003 - accuracy: 0.2062
Epoch 8/30
33/33 [==============================] - 0s 3ms/step - loss: 3.2416 - accuracy: 0.2785
Epoch 9/30
33/33 [==============================] - 0s 3ms/step - loss: 2.8852 - accuracy: 0.4050
Epoch 10/30
33/33 [==============================] - 0s 3ms/step - loss: 2.5907 - accuracy: 0.4503
Epoch 11/30
33/33 [==============================] - 0s 3ms/step - loss: 2.3070 - accuracy: 0.5034
Epoch 12/30
33/33 [==============================] - 0s 3ms/step - loss: 2.1093 - accuracy: 0.5259
Epoch 13/30
33/33 [==============================] - 0s 3ms/step - loss: 1.8081 - accuracy: 0.6421
Epoch 14/30
33/33 [==============================] - 0s 3ms/step - loss: 1.6692 - accuracy: 0.6611
Epoch 15/30
33/33 [==============================] - 0s 3ms/step - loss: 1.5044 - accuracy: 0.7138
Epoch 16/30
33/33 [==============================] - 0s 3ms/step - loss: 1.3213 - accuracy: 0.7485
Epoch 17/30
33/33 [==============================] - 0s 3ms/step - loss: 1.2146 - accuracy: 0.7702
Epoch 18/30
33/33 [==============================] - 0s 3ms/step - loss: 1.1265 - accuracy: 0.7707
Epoch 19/30
33/33 [==============================] - 0s 3ms/step - loss: 1.0800 - accuracy: 0.7852
Epoch 20/30
33/33 [==============================] - 0s 3ms/step - loss: 0.9658 - accuracy: 0.8111
Epoch 21/30
33/33 [==============================] - 0s 3ms/step - loss: 0.8986 - accuracy: 0.8288
Epoch 22/30
33/33 [==============================] - 0s 3ms/step - loss: 0.8423 - accuracy: 0.8434
Epoch 23/30
33/33 [==============================] - 0s 3ms/step - loss: 0.7342 - accuracy: 0.8816
Epoch 24/30
33/33 [==============================] - 0s 3ms/step - loss: 0.6530 - accuracy: 0.8965
Epoch 25/30
33/33 [==============================] - 0s 3ms/step - loss: 0.6405 - accuracy: 0.8835
Epoch 26/30
33/33 [==============================] - 0s 3ms/step - loss: 0.5748 - accuracy: 0.9099
Epoch 27/30
33/33 [==============================] - 0s 3ms/step - loss: 0.5470 - accuracy: 0.9244
Epoch 28/30
33/33 [==============================] - 0s 3ms/step - loss: 0.5260 - accuracy: 0.9134
```

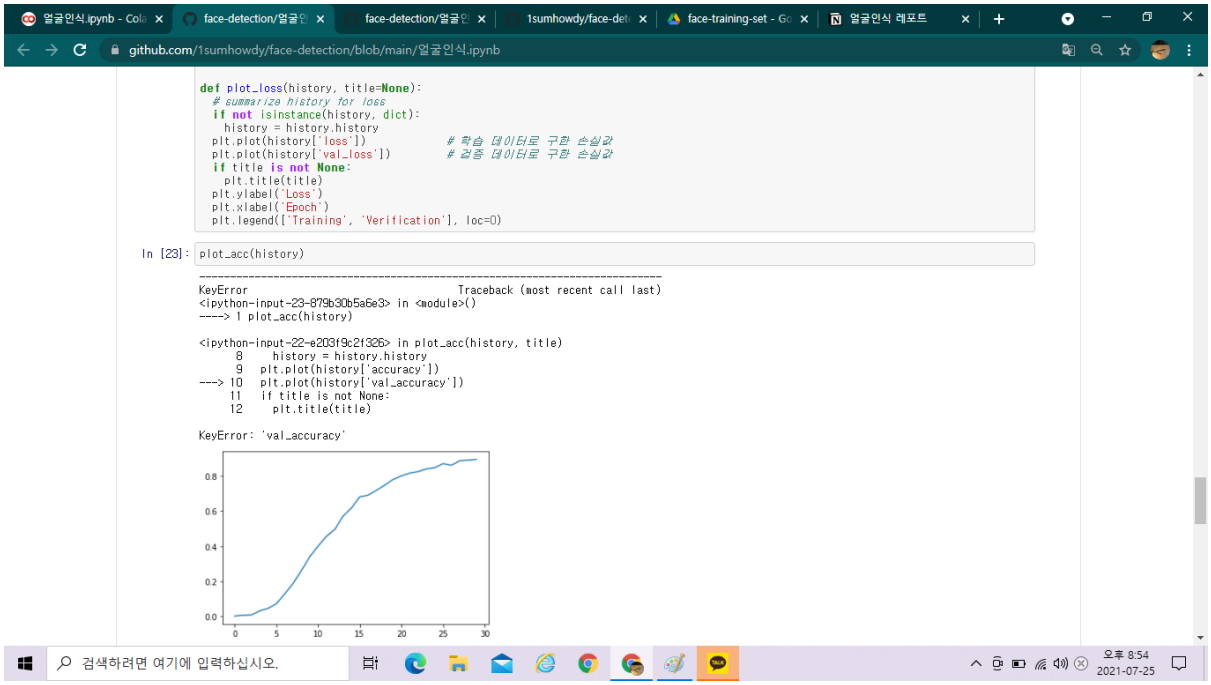
```
Epoch 29/30
33/33 [==============================] - 0s 3ms/step - loss: 0.4873 - accuracy: 0.9296
Epoch 30/30
33/33 [==============================] - 0s 3ms/step - loss: 0.4413 - accuracy: 0.9234
```

```
import matplotlib.pyplot as plt

def plot_acc(history, title=None):
  # summarize history for accuracy
  if not isinstance(history, dict):
    history = history.history
  plt.plot(history['accuracy'])
  plt.plot(history['val_accuracy'])
  if title is not None:
    plt.title(title)
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Training', 'Verification'], loc=0)   # 두 선의 이름(Train, Test) 표시

def plot_loss(history, title=None):
  # summarize history for loss
  if not isinstance(history, dict):
    history = history.history
  plt.plot(history['loss'])             # 학습 데이터로 구한 손실값
  plt.plot(history['val_loss'])         # 검증 데이터로 구한 손실값
  if title is not None:
    plt.title(title)
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Training', 'Verification'], loc=0)
```

학습 결과를 분석하기 위해 그래프를 구현했다.



epoch 30/30, 정확도=0.92. 높은 정확도가 나타난다.

```
x_predict = model.predict(x_train)
y_predict = []

print(x_predict.shape)

for image in x_predict:
    y_predict.append(np.argmax(image))

y_predict = np.array(y_predict)
```

x_train=이미지, 문제

y_train=인덱스, 답안 (y_predict = [ ] 인 이유)

```
def accuracy(original, x_predict):
    accuracy = original == x_predict
    accuracy = np.count_nonzero(accuracy)

    return accuracy / original.shape[0]
```

정확도 출력 함수

```
accuracy(y_train, y_predict)
```

모델에 학습시켰던 것을 바탕으로, x_train 입력시 출력된 y_predict를 y_train과 비교하여 모델의 정확도 출력

→ 0.9552380952380952

높은 정확도가 나타난다.

```
# test
predictions = model.predict(x_test)
print(predictions.shape)

y_test = []
for image in predictions:
    y_test.append(np.argmax(image))

y_test = np.array(y_test)

#결과
print(y_test)
```

test데이터의 y-test 예측하기

```
(700, 351)
[118 319 137 221  51 153 132 326 309 212 215 224 212 177 227 345 221 239
 271 180 225 216 158 263 269 284 251 334  55 324 266  34 287 194 280 188
 248  34  34 319  34 177 279 241 342 283 186 125 155 164 229 238 238 118
 128 162 168 303 291 238 279 347 218 213 192 101 226 154 202  98 225 240
 314 181 126 290 220 275 294 115 224 101 236  77  91 199 160  90  61 239
 187  87 188 137 254 144 308 118  93 226 116 167 238 169 288 229 314 238
 196   3 334 146 155 270  92 320  30 218 334 276 203 205 217 151  77 345
 334 257 172 220 279 262 267 313  93 165 178 187 248 128  34 323 340 212
  88 234 235 201 345  85 118 221 235 212 213 236 317 264 316 334 334 328
 213  39  77 229  85 162 191 202 101 183  97  47 278 192 314 334 168 276
 218  86 225 226 243 155 155 263 334 239 312 285  38 296 229 187 215 257
 104 324 165  76 253 223 190 334 305 239 221 239  64 157 229 284 248 193
  92 283 151 220  93 271 161 280  77 213 343 345 218 312 194 185 252 235
 175 250 334 221  34 327 168 239 334 122 261 213 323 118 325 308 156 198
 303 303 307 121 282 182 344 212  64 334 238  95 163  86  86 172 323 175
 159 303  87 238 321 238 349 275 105 244 264 186 301 130 263 157 328 234
 316 232 174  88 334 212 129  79 218 164  34 320 215 137 300 204 334 112
 312 294 236 328  34  86 172 199 334 215 213  96  86  96 225 248 168 185
 129 157 226 213 226  34 163 172 137 303 214 225  86 275 153 323 334 310
 228 238 207 238 317 288 211 334 239 212 121 320 133 268 172 297 295 151
 290 239 226 213 213  21 260  99 314 245 150 239 212 172 270 132 118 334
  47 253 231 118 297 340  89 314  34 213 262  97 310 283 214 159 262 246
  76 128 335 242 304 215 137  64 348  76 267 209 327 263 173 320  85 227
 346 220 345 197 259 284 218 198  34 304 285 324 212 295 238 278 248  11
 212 239  76  39  34 159 334 314  45 213  92 270 230 157 214 261 136 246
 225 303 224 212 315 248 221 133 207 281 136 227 345  34 334 322 146 172
 220 247 239 228 269 252 111  56 334 172 281 283 244 172 117 107 298  96
 312 213 216 250 213 213 110 347 239 161 345 227 286 258 345 112  47  97
 300 217 176 163 152 238 214 154  92 208 197 131 231 268 244  34 185 257
 334  21 126  61 339 125 110 314 259 256 213 288 177  92 152 250 220 221
```

```
 156 222  34 229 229 322 225 255 192  34 241 212 223 325 196 303 334 209
 100 242 189  47  27 320 158 238 208  81 334 173 201 184 334 341  30 334
 117 166 275 122 340 157 226 239 218 165 149 299 350 343 228 238 313 212
 204 106 160 152 269 266 212 314 213 181 239 254  80 201 244 183 338 205
 288 155 275 118 192 306 185  40 226 321 227 228 218 201  76 239 343 311
 246  57 115 157   1 303 213 166 303 239 345  84  86 199 323 169 345 185
 174 300 240 222 182  16  93 267 253 136 345 244 339 273 334 226 269 227
 225 207  92 320 266 336 334 112 281 224 324 152 221 220 128  48 239 338
 123 220 149  93 347 128 261 225 210 238 212  30   8 309 157 226]
```