생체인식보안

홍채 인식 #2

사이버보안전공 1871095 문수빈

01. Training Data

01. 1차 시도와의 차이

```
# Image Augmentation 생성
datagen = ImageDataGenerator(
            width_shift_range = 0.01,
            height_shift_range = 0.01,
            horizontal_flip = True,
            vertical_flip = True,
            zoom_range = 0.05,
            brightness_range=[1.0, 1.0])
```

1. 다양한 Image Augmentation 효과 적용

처음에는 이미지 증강으로 생성된 데이터가 너무 달라지면 학습이 잘 이루어지지 않을 것이라고 생각해, width_shift와 height_shift만 수행했는데 오히려 이 경우 데이터가 사진 영역의 중심에서 벗어나 학습이 잘 이루어지지 않는다는 것을 깨달음. width_shift와 height_shift의 값을 상대적으로 줄이고, 밝기 조절과 좌우 및 상하 반전의 효과를 넣어 Image Augmentation을 진행.

```
def build_model():

    input = Input(shape = (IMAGE_HEIGHT, IMAGE_WIDTH, 3))
    base_model = ResNet50V2(include_top = False, weights = 'imagenet', input_tensor = input)

    for layer in base_model.layers[:44]:
      layer.trainable = False

    X = base_model.output
    X = BatchNormalization()(X)
    X = GlobalAveragePooling2D()(X)
    X = Dense(512, activation = 'relu')(X)
    X = Dropout(0.5)(X)

    Y = Dense(64, activation='softmax')(X)

    model = tf.keras.Model(inputs = input, outputs = Y)

    model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.0001), metrics=['accuracy',Precision(),Recall()])
    model.summary()
```

2. ResNetV2 Pre-trained Model 사용

1차 때는 CNN을 이용해 모델을 생성했는데, 성능이 그리 좋지 않아 ResNetV2로 학습을 진행.

freezing layer는 1차 발표 때 성능이 좋았다고 얘기된 44를 사용했다.

01. Training Data

01. 1차 시도와의 차이

3. HoughCircle의 삭제

전처리된 데이터가 중요할 것이라 생각해 정확히 홍채 영역만 크롭한 데이터를 생성하고자 했는데,

해당 방식의 전처리가 그렇게 중요하지 않다는 것을 알았다.

또한 데이터의 Accuracy 차이가 있을까 싶어서 앞서 말한 ResNetV2 모델로 홍채 영역만 전처리된 모델을 학습시켰지만

일반 데이터보다도 성능이 매우 좋지 않았다.

```
Epoch 59/60
8/8 [==============================] - 5s 579ms/step - loss: 0.1587 - accuracy: 0.9421 - precision_2: 0.9893 - recall_2: 0.8719
Epoch 60/60
8/8 [==============================] - 5s 577ms/step - loss: 0.1539 - accuracy: 0.9452 - precision_2: 0.9893 - recall_2: 0.8721
Epoch 1/60
8/8 [==============================] - 5s 576ms/step - loss: 0.2481 - accuracy: 0.9208 - precision_2: 0.9893 - recall_2: 0.8723
Epoch 2/60
8/8 [==============================] - 5s 573ms/step - loss: 0.2438 - accuracy: 0.9198 - precision_2: 0.9892 - recall_2: 0.8724
Epoch 3/60
8/8 [==============================] - 5s 576ms/step - loss: 0.2335 - accuracy: 0.9198 - precision_2: 0.9891 - recall_2: 0.8725
Epoch 4/60
8/8 [==============================] - 5s 576ms/step - loss: 0.2601 - accuracy: 0.9157 - precision_2: 0.9890 - recall_2: 0.8726
Epoch 5/60
8/8 [==============================] - 5s 576ms/step - loss: 0.2323 - accuracy: 0.9208 - precision_2: 0.9889 - recall_2: 0.8727
Epoch 6/60
8/8 [==============================] - 5s 577ms/step - loss: 0.1778 - accuracy: 0.9411 - precision_2: 0.9889 - recall_2: 0.8728
Epoch 7/60
8/8 [==============================] - 5s 577ms/step - loss: 0.1773 - accuracy: 0.9360 - precision_2: 0.9889 - recall_2: 0.8730
Epoch 8/60
8/8 [==============================] - 5s 577ms/step - loss: 0.1919 - accuracy: 0.9299 - precision_2: 0.9888 - recall_2: 0.8732
```

=> 특정 구간부터는 Accuracy가 전혀 증가하지 않았다.

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 2.4132652282714844
Test accuracy: 0.673758864402771
```

=> Test dataset의 Accuracy가 0.6으로 낮았다.

02. Build Model

02. 모델 설계

```python
def build_model():

    input = Input(shape = (IMAGE_HEIGHT, IMAGE_WIDTH, 3))
    base_model = ResNet50V2(include_top = False, weights = 'imagenet', input_tensor = input)

    for layer in base_model.layers[:44]:
      layer.trainable = False

    X = base_model.output
    X = BatchNormalization()(X)
    X = GlobalAveragePooling2D()(X)
    X = Dense(512, activation = 'relu')(X)
    X = Dropout(0.5)(X)

    Y = Dense(64, activation='softmax')(X)

    model = tf.keras.Model(inputs = input, outputs = Y)

    model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.0001), metrics=['accuracy',Precision(),Recall()])
    model.summary()
```

```python
model = build_model()
model.summary()
```

```
Model: "model_10"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
Total params: 24,654,912
Trainable params: 24,346,432
Non-trainable params: 308,480
_________________________________________________________________________________________________
Model: "model_10"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
Total params: 24,654,912
Trainable params: 24,346,432
Non-trainable params: 308,480
_________________________________________________________________________________________________
```

모델은 ResNetV2를 사용하였다.

Input으로 (IMAGE_HEIGHT, IMAGE_WIDTH, CHANNEL(3)) 값을 넣었으며, Freezing Layer로는 44를 사용.

손실 함수는 categorical_crossentropy이며 learning rate = 0.0001이다.

01. Load Image

02. 데이터 학습

```
# 이미지 훈련을 3회로 분할해 교차검증 진행
skf = StratifiedKFold(n_splits=3)
```

```
#모델을 사용한 훈련 진행

all_history = []
all_acc = []
all_f1 = []
all_recall = []
all_precision = []

for train_index, test_index in skf.split(X_data, Y_data):
    X_train, X_test, Y_train, Y_test = train_test_split(X_data,
                                                        Y_data,
                                                        test_size = 0.3,
                                                        random_state=10)

    X_train = np.array(X_train)
    X_test = np.array(X_test)

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    Y_train = to_categorical(Y_train, nb_classes)
    Y_test = to_categorical(Y_test, nb_classes)

    X_train /= 255
    X_test /= 255

    history = model.fit(X_train, Y_train, epochs = 60, batch_size = 128, verbose = 1)
    all_history.append(history.history)

    Y_pred = model.predict(X_test)
    Y_pred_binary = np.around(Y_pred)

    acc = accuracy_score(Y_test, Y_pred_binary)
    f1 = f1_score(Y_test, Y_pred_binary, average = 'micro')
    recall = recall_score(Y_test, Y_pred_binary, average = 'micro')
    precision = precision_score(Y_test, Y_pred_binary, average = 'micro')

    all_acc.append(acc)
    all_f1.append(f1)
    all_recall.append(recall)
    all_precision.append(precision)
```

트레이닝 : 테스트 = 7 : 3

데이터 형변환

데이터 정규화 과정

K_fold cross validation을 처음에는 5회 적용했으나 ResNet 모델로 변경 후 validation을 3회만 진행해도 1.0에 가까운 accuracy가 나와 3회로 감소시켰다.

epochs	60
batch_size	128
KFold	3

```
Epoch 6/60
8/8 [==============================] - 8s 1s/step - loss: 0.5860 - accuracy: 0.8284 - precision: 0.8907 - recall: 0.1234
Epoch 7/60
8/8 [==============================] - 8s 1s/step - loss: 0.4323 - accuracy: 0.8569 - precision: 0.8992 - recall: 0.2206
Epoch 8/60
8/8 [==============================] - 8s 1s/step - loss: 0.1504 - accuracy: 0.9624 - precision: 0.9175 - recall: 0.3081
Epoch 9/60
8/8 [==============================] - 8s 1s/step - loss: 0.0557 - accuracy: 0.9848 - precision: 0.9356 - recall: 0.3848
Epoch 10/60
8/8 [==============================] - 8s 1s/step - loss: 0.0296 - accuracy: 0.9888 - precision: 0.9474 - recall: 0.4468
Epoch 11/60
8/8 [==============================] - 8s 1s/step - loss: 0.0160 - accuracy: 0.9970 - precision: 0.9564 - recall: 0.4980
Epoch 12/60
8/8 [==============================] - 8s 1s/step - loss: 0.0078 - accuracy: 0.9980 - precision: 0.9628 - recall: 0.5408
Epoch 13/60
8/8 [==============================] - 8s 1s/step - loss: 0.0085 - accuracy: 0.9970 - precision: 0.9674 - recall: 0.5767
Epoch 14/60
8/8 [==============================] - 8s 1s/step - loss: 0.0055 - accuracy: 0.9990 - precision: 0.9709 - recall: 0.6075
Epoch 15/60
8/8 [==============================] - 8s 1s/step - loss: 0.0026 - accuracy: 1.0000 - precision: 0.9739 - recall: 0.6341
Epoch 16/60
8/8 [==============================] - 9s 1s/step - loss: 0.0014 - accuracy: 1.0000 - precision: 0.9764 - recall: 0.6574
```

03. Training Data

교차 검증별 평균 스코어

```
print(sum(all_acc)/3)
print(sum(all_f1)/3)
print(sum(all_recall)/3)
print(sum(all_precision)/3)
```

Accuracy	0.99
F1_score	0.99
Recall	0.99
Precision	0.99

최종 스코어

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

Test score	0.012
Test accuracy	0.99

최종 데이터 훈련 결과 모델의 테스트 스코어는 0.0129로, 테스트의 정확도는 0.99로 확인되었다.

03. Result

01. 테스트 데이터 전처리

```
test_dataset = []
tmp = []

for filename in order_list:
    imagePath = path + '/03_iris_test/' + filename

    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT))

    tmp.append(filename)
    test_dataset.append(image)
```

```
test_dataset = np.array(test_dataset)
test_dataset = test_dataset.astype('float32')
test_dataset /= 255
predict = model.predict(test_dataset).argmax(1)
```

주어진 테스트 데이터도 전처리한 후, 모델을 예측한다.

```
result = pd.DataFrame({'Image':[],
                       'Answer':[]})
```

```
for i in range(len(predict)):
  new_data = {'Image':int(i+1), 'Answer':int(predict[i])+1}
  result = result.append(new_data, ignore_index = True)
```

```
result
```

	Image	Answer
0	1.0	46.0
1	2.0	59.0
2	3.0	26.0
3	4.0	46.0
4	5.0	51.0
...	...	...
251	252.0	21.0
252	253.0	14.0
253	254.0	26.0
254	255.0	45.0
255	256.0	16.0

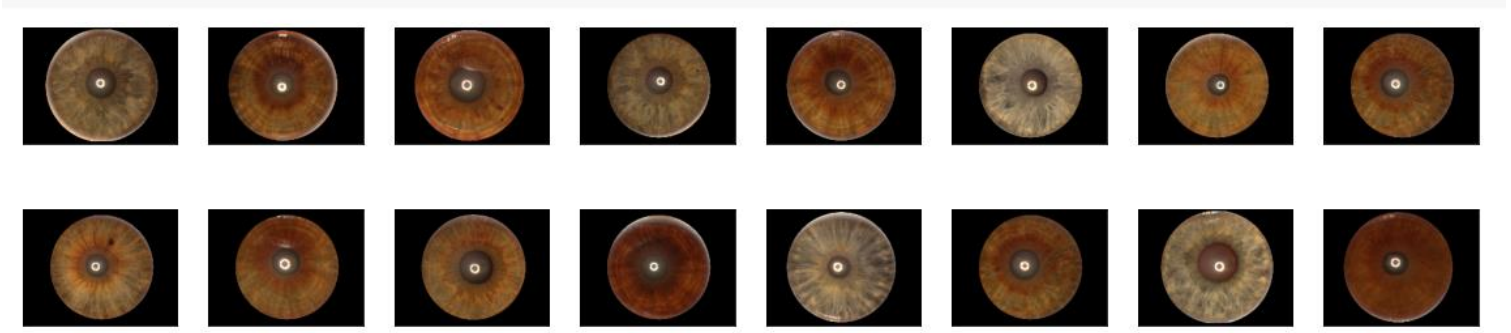
256 rows × 2 columns

predict 값을 DataFrame에 넣어 결과를 출력한다.

04. Evaluation

다양한 시도와 평가

1. ResNet 모델의 Learning rate를 0.001로 했더니 Accuracy가 늘어나다가도 갑자기 한 구간에서 더 이상 값이 변화하지 않고 멈춰있어 learning rate를 0.0001로 감소시켰더니 학습이 제대로 이루어지기 시작했다.
2. 홍채 데이터를 cv2_color.BGR2GRAY로 변환하고 houghcircle을 적용해 데이터 학습이 제대로 이루어지지 않나 싶어 BGR2RGB로 변환해 시도해보았으나 여전히 학습이 이루어지지 않았다.



3. RAM이 부족해 이미지 사이즈를 줄이면 학습 결과가 더 안 좋아져서, image /= 255.로 픽셀 정규화 시킨 후 이미지 사이즈를 키워 학습 성능을 높일 수 있었다.
4. 처음 numpy로 되어 있는 데이터를 정규화시켜 학습 했을 때는 정확도가 0.89 정도로 나왔다. numpy 트레이닝 데이터를 X_train.astype('float32')로 형을 변환한 후 정규화를 시켜 학습했을 때 갑자기 0.99로 테스트셋의 정확도가 증가했다. 데이터 값이 정밀해져서 그런 것인지 결과 향상의 확실한 이유를 모르겠다.

04. Evaluation

다양한 시도와 평가

CNN 모델을 사용할 때보다 ResNet Pre-trained 모델을 사용했을 때 성능이 눈에 띄게 향상되었다.
그간의 과제에서 KNN, SVM, CNN 등 고전적인 방식만 사용했는데 사용할 수 있는 더 다양한 모델이 있음을 알았다.

HoughCircle을 이용한 전처리는 오히려 성능이 안 좋았기 때문에 제거했으나
대부분의 트레이닝과 테스트셋 이미지가 유사해 별도의 전처리 없이도 모델이 홍채를 잘 인식하지 않았을까 한다.
그러나 실제 홍채 인식을 적용할 때는 다른 전처리 및 정규화 방식을 이용해 모델을 전처리해야
외부의 영향 없이 홍채를 잘 분류하는 결과를 낼 수 있을 듯 하다.