생체인식보안

멀티모달 #1 Model 1

사이버보안전공 1871095 문수빈

01. Preprocessing Data

01. 데이터 전처리

```
data = []
label= []
dtype = []

for filename in train_order:
    category = filename[:3]
    imagePath = path + '/04_multimodal_training/' + filename
    if filename[4:8] == 'face':
      dtype.append('F')
    elif filename[4:8] == 'iris':
      dtype.append('I')
```

이미지를 불러와 각각 얼굴과 홍채로 타입을 나누어 저장.

한 이미지 당 15개의 image augmenation을 진행하여, 한 라벨에 총 16개의 데이터를 만들었다.

```
# iris와 face 각각 훈련데이터셋 형성

face_X = []
face_Y = []
iris_X = []
iris_Y = []

for i in range((len(data))):
  if dtype[i] == 'F':
    face_X.append(data[i])
    face_Y.append(label[i])
  elif dtype[i] == 'I':
    iris_X.append(data[i])
    iris_Y.append(label[i])
```

학습시킬 데이터를 face_X, iris_X로 생성하고, 라벨을 face_Y, iris_Y로 생성한다.

02. Build Model

02. 1차 시도

홍채 인식

```
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()
```

홍채 인식 모델 - ResNetV2

얼굴 인식

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(4096, (7, 7), activation='relu'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), activation='relu'))
model.add(Dropout(0.5))
model.add(Convolution2D(2622, (1, 1)))
model.add(Flatten())
model.add(Activation('softmax'))
```

얼굴 인식 모델 - vgg16

홍채와 얼굴 데이터를 따로 학습시켜 예측 결과를 도출.

이후 예측결과가 다르다면 테스트 데이터셋의 정확도가 더 높은 모델의 결과를 최종 결과로 선정.

02. Build Model

02. 2차 시도

```
# iris와 face 각각 훈련데이터셋 형성

face_X = []
face_Y = []
iris_X = []
iris_Y = []

for i in range((len(data))):
  tmp = []
  if dtype[i] == 'F':
    face_X.append(data[i])
    face_Y.append(label[i])
  elif dtype[i] == 'I':
    iris_X.append(data[i])
    iris_Y.append(label[i])
```

CNN으로 각각 데이터 학습을 한 이후 SVM으로 분류할 경우 모델 1개를 사용한 경우라 생각했으나
1차 발표 당시 피드백을 듣고 1개 모델 사용의 경우,
홍채와 얼굴 데이터를 처음부터 합쳐 ResNetV2로 훈련 데이터를 학습시키는 방향으로 변경.

```
# 두 데이터셋을 짝지어 하나로 합침

train_X = []
train_Y = []

i = 0
for i in range(len(face_X)):
  data[i]
  train_X.append(face_X[i]+iris_X[i])
  train_Y.append(face_Y[i])
```

face_X와 iris_X를 합쳐 하나의 train_X 데이터 생성

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
_________________________________________________________________________________________________
```

인식 모델은 ResNetV2를 사용하였다.

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

손실 함수는 categorical_crossentropy이며 learning rate = 0.0001이다.

03. Training Data

03. 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

3 K_fold cross validation을 진행하였다.

epochs	60
batch_size	128
KFold	3

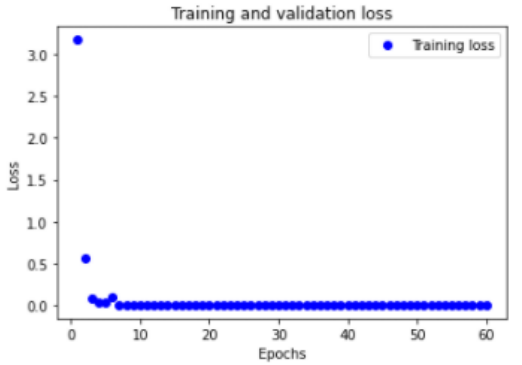
```
Epoch 2/60
23/23 [==============================] - 17s 732ms/step - loss: 0.5543 - accuracy: 0.8409 - precision: 0.9041 - recall: 0.2730
Epoch 3/60
23/23 [==============================] - 17s 745ms/step - loss: 0.0831 - accuracy: 0.9791 - precision: 0.9428 - recall: 0.5282
Epoch 4/60
23/23 [==============================] - 17s 757ms/step - loss: 0.0373 - accuracy: 0.9892 - precision: 0.9621 - recall: 0.6588
Epoch 5/60
23/23 [==============================] - 18s 778ms/step - loss: 0.0241 - accuracy: 0.9948 - precision: 0.9707 - recall: 0.7321
Epoch 6/60
23/23 [==============================] - 18s 779ms/step - loss: 0.0917 - accuracy: 0.9850 - precision: 0.9757 - recall: 0.7790
Epoch 7/60
23/23 [==============================] - 18s 772ms/step - loss: 0.0063 - accuracy: 0.9997 - precision: 0.9790 - recall: 0.8115
Epoch 8/60
23/23 [==============================] - 18s 791ms/step - loss: 0.0013 - accuracy: 0.9993 - precision: 0.9823 - recall: 0.8366
Epoch 9/60
23/23 [==============================] - 19s 823ms/step - loss: 0.0013 - accuracy: 0.9993 - precision: 0.9846 - recall: 0.8557
Epoch 10/60
23/23 [==============================] - 19s 832ms/step - loss: 1.7217e-04 - accuracy: 1.0000 - precision: 0.9864 - recall: 0.8708
```

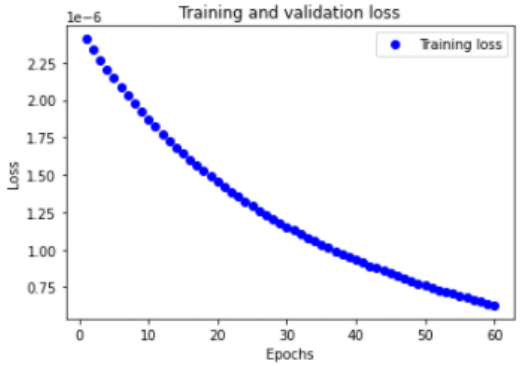
03. Training Data

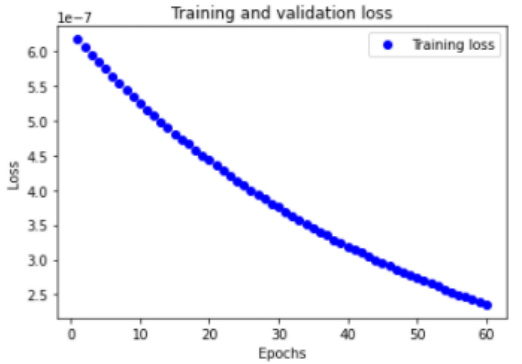
03. 데이터 학습

훈련 단계 별 Training and validation loss 이다.

```
for i in range(3):
    loss =all_history[i]['loss']
    #val_loss = all_history[i]['val_loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```







학습 도중 loss 값의 큰 변화 없이 지속적으로

학습이 잘 이루어지는 것을 볼 수 있다.

03. Training Data

교차검증 스코어

```
# accuracy

accuracy = sum(all_acc)/3

print('Average accuracy : ', accuracy)
print('Average recall : ', sum(all_recall)/3)
print('Average F1 score : ', sum(all_f1)/3)
print('Average precision : ', sum(all_precision)/3)
```

```
Average accuracy :  0.9951179820992677
Average recall :  0.9951179820992677
Average F1 score :  0.995928338762215
Average precision :  0.9967400162999184
```

Accuracy	0.99
F1_score	0.99
Recall	0.99
Precision	0.99

테스트 데이터 스코어

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

```
Test score: 0.01583733782172203
Test accuracy: 0.9951179623603821
```

Accuracy	0.99
Score	0.015

04. Test Data

04. 테스트 데이터 전처리

```
# face와 test 각각 배열에 저장
face_test = []
iris_test = []

for filename in order_list:
    imagePath = path + '/04_multimodal_test/' + filename

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

    i = 0
    if filename[-8:-4] == 'face':
      new_image = (image / 255.).astype(np.float32)
      new_image = cv2.resize(new_image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
      face_test.append(face_descriptor.predict(np.expand_dims(new_image, axis=0))[0])
    elif filename[-8:-4] == 'iris':
      iris_test.append(image)
```

주어진 테스트 데이터에서 얼굴과 홍채를 각 배열에 저장하고,

두 결과를 하나의 test_X 배열로 합친다.

```
test_X = []

for i in range(len(face_test)):
  test_X.append(face_test[i]+iris_test[i])
```

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

```
# model result
pred
```

```
array([27, 31, 22, 57, 54, 45,  5, 12,  8,  7, 36,  2, 46, 50, 10, 61, 49,
       59, 56, 13, 37, 30, 25,  0, 41, 58,  9, 20, 48, 47, 42, 24,  3,  6,
       55, 63, 21, 16, 34, 11, 35, 32, 40, 62, 19, 52, 33, 15,  1, 29, 17,
       26, 53, 28, 18, 14, 43, 60,  4, 39, 38, 23, 51, 44])
```

model.predict를 통해 데이터의 분류값을 예측한다.

04. Result

04. 결과 생성

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

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

```
result
```

	Image	Answer
0	0.0	27.0
1	1.0	31.0
2	2.0	22.0
3	3.0	57.0
4	4.0	54.0
...	...	...
59	59.0	39.0
60	60.0	38.0
61	61.0	23.0
62	62.0	51.0
63	63.0	44.0

64 rows × 2 columns

최종 결과를 csv 파일로 export한다.

05. Evaluation

05. 평가

- ResNetV2 모델의 경우 높은 epoch와 많은 cross validation이 아니어도 학습 데이터의 accuracy 증가 속도가 굉장히 빠른 것을 볼 수 있었다.
- ResNetV2로 단일 홍채 인식만 시도할 때보다 더 높은 정확도 결과를 얻을 수 있었으며,
- 실제 생체 인증에 단일 모델보다 멀티 인증을 사용할 경우 더 정확한 식별 결과를 얻을 수 있을 것 같다.

생체인식보안

멀티모달 #1 Model 2

사이버보안전공 1871095 문수빈

01. Preprocessing Data

01. 데이터 전처리

```
data = []
label= []
dtype = []

for filename in train_order:
    category = filename[:3]
    imagePath = path + '/04_multimodal_training/' + filename
    if filename[4:8] == 'face':
      dtype.append('F')
    elif filename[4:8] == 'iris':
      dtype.append('I')
```

이미지를 불러와 각각 얼굴과 홍채로 타입을 나누어 저장.

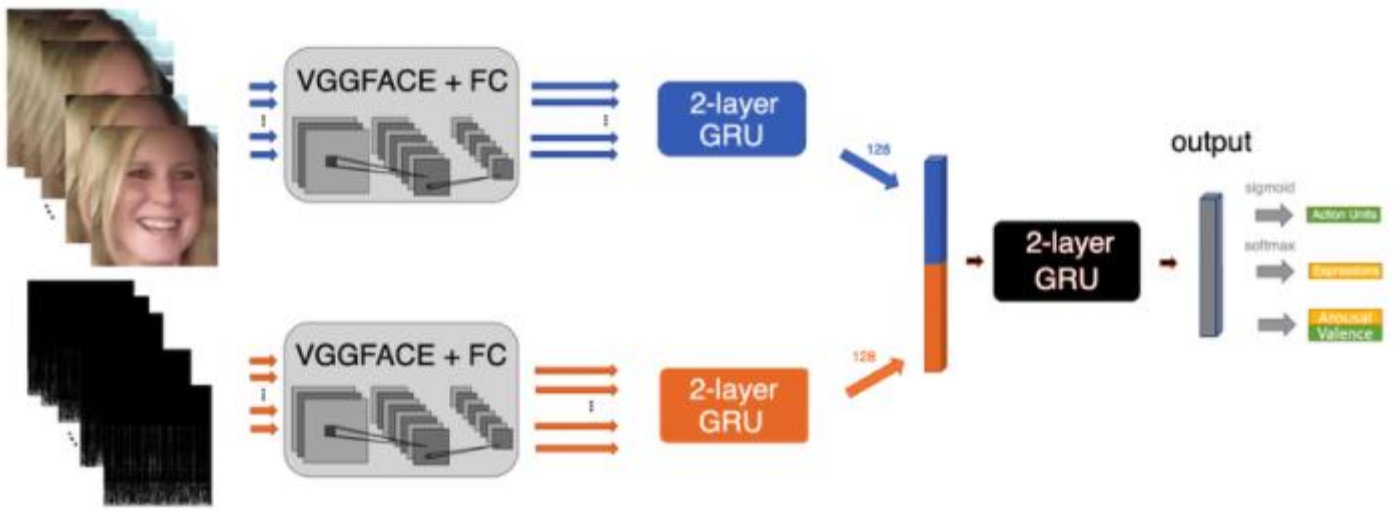
한 이미지 당 5개의 image augmenation을 진행하여, 한 라벨에 총 6개의 데이터를 만들었다.

```
# iris와 face 각각 훈련데이터셋 형성

face_X = []
face_Y = []
iris_X = []
iris_Y = []

for i in range((len(data))):
  if dtype[i] == 'F':
    face_X.append(data[i])
    face_Y.append(label[i])
  elif dtype[i] == 'I':
    iris_X.append(data[i])
    iris_Y.append(label[i])
```

학습시킬 데이터를 face_X, iris_X로 생성하고, 라벨을 iris_X, iris_Y로 생성한다.

02. Build Model

02. 멀티모달 모델 생성



기학습된 CNN 모델인 vgg를 이용해 얼굴과 홍채 각각 특징을 검출하고, 나온 특징을 합쳐 SVM으로 분류해 최종 결과를 생성한다.

02. Build Model

02. CNN 모델 형성

트레이닝 데이터에서 특징점 추출을 위해 VGGNet16 모델을 사용한다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d (ZeroPadding2 (None, 226, 226, 3)       0
_________________________________________________________________
```

...

```
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 1, 1, 4096)        16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1, 4096)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 1, 1, 2622)        10742334
_________________________________________________________________
flatten (Flatten)            (None, 2622)              0
_________________________________________________________________
activation (Activation)      (None, 2622)              0
=================================================================
Total params: 145,002,878
Trainable params: 145,002,878
Non-trainable params: 0
```

```
model = build_model()
model.load_weights(path+'/vgg_face_weights.h5')
model.summary()
```

모델은 총 16개의 CNN 복합 레이어로 구성되어 있으며,

'vgg_face_weights.h5' 파일을 이용해 모델 학습에 가중치를 더한다.

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(4096, (7, 7), activation='relu'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), activation='relu'))
model.add(Dropout(0.5))
model.add(Convolution2D(2622, (1, 1)))
model.add(Flatten())
model.add(Activation('softmax'))
```

전체 CNN 모델

01. Training Data

02. 특징 추출 모델 형성

```
from tensorflow.keras.models import Model
descriptor = Model(inputs=model.layers[0].input, outputs=model.layers[-2].output)
```

CNN 기반의 VGGNet16 모델을 이용해 학습을 진행했다.

기존 CNN 레이어층에 기학습된 'vgg_face_weights' 가중치를 더하고 model의 input과 output을 연결해 descriptor 모델을 형성한다.

```
# 이미지를 64개의 카테고리로 분류
nb_classes = 64

face_embeddings = np.zeros((nb_classes*24, 2622))

i = 0

for image in face_X:
    image = (image / 255.).astype(np.float32)
    image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    # descriptor 모델에서 나온 특징점들을 저장
    face_embeddings[i]= descriptor.predict(np.expand_dims(image, axis=0))[0]
    i += 1
```

```
# 이미지를 64개의 카테고리로 분류
nb_classes = 64

iris_embeddings = np.zeros((nb_classes*24, 2622))

i = 0

for image in iris_X:
    image = (image / 255.).astype(np.float32)
    image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)
    # face_descriptor 모델에서 나온 특징점들을 저장
    iris_embeddings[i]= descriptor.predict(np.expand_dims(image, axis=0))[0]
    i += 1
```

모델을 이용해 face 데이터와 iris 데이터의 특징점을 추출한다.

03. Training data

03. 데이터 트레이닝

```
# CNN으로 예측한 홍채와 얼굴 특징점 데이터를 하나의 특징으로 합침
embeddings = []

i = 0
for i in range(len(iris_embeddings)):
  embeddings.append(face_embeddings[i] + iris_embeddings[i])
  i += 1
```

VGGNet으로 예측한 홍채와 얼굴의 특징점 데이터를 하나의 특징으로 합친다.

```
# 특징 데이터들을 train과 test로 분할
# 4개의 데이터마다 하나씩 테스트 데이터로 사용
train_idx = np.arange(nb_classes*24) % 4 != 0
test_idx = np.arange(nb_classes*24) % 4 == 0

X_train = embeddings[train_idx]
X_test = embeddings[test_idx]
Y_train = face_Y[train_idx]
Y_test = face_Y[test_idx]
```

추출한 특징점들을 train data로, 그 중에서 4개의 데이터마다 하나씩 테스트 데이터로 사용한다.

```
# SVM으로 학습 데이터 훈련
svc = LinearSVC(C = 10)
svc.fit(X_train, Y_train)
y_pred = svc.predict(X_train)

# 3번의 cross validation을 통해 training set score 교차 검증
print("Accuracy on training set: ", cross_val_score(svc, X_train, Y_train, cv=3))
print("Accuracy on test set: ", svc.score(X_test, Y_test))
```

```
Accuracy on training set:  [0.99479167 0.9921875  0.97135417]
Accuracy on test set:  1.0
```

특징은 SVM을 이용해 학습하며, 분류한다.

3 K-fold cross validation을 사용해 교차 검증한다.

03. Training data

03. 데이터 트레이닝

얼굴 모델만 학습했을 때

```
# SVM으로 학습 데이터 훈련
svc = LinearSVC(C = 10)
svc.fit(face_X_train, face_Y_train)
face_pred = svc.predict(face_X_train)

# 3번의 cross validation을 통해 training set score 교차 검증
print("Accuracy on training set: ", cross_val_score(svc, face_X_train, face_Y_train, cv=3))
print("Accuracy on test set: ", svc.score(face_X_test, face_Y_test))
```

```
Accuracy on training set:  [0.99739583 0.984375   0.95833333]
Accuracy on test set:  1.0
```

홍채 모델만 학습했을 때

```
# SVM으로 학습 데이터 훈련
svc = LinearSVC(C = 10)
svc.fit(iris_X_train, iris_Y_train)
iris_pred = svc.predict(iris_X_train)

# 3번의 cross validation을 통해 training set score 교차 검증
print("Accuracy on training set: ", cross_val_score(svc, iris_X_train, iris_Y_train, cv=5))
print("Accuracy on test set: ", svc.score(iris_X_test, iris_Y_test))
```

```
Accuracy on training set:  [0.88311688 0.86147186 0.89565217 0.88695652 0.66521739]
Accuracy on test set:  0.9505208333333334
```

단일 얼굴, 단일 홍채만 분류했을 때보다 두 모델을 모두 이용해 결과를 냈을 때 결과가 더 정확했다.

03. Training Data

교차 검증별 평균 스코어

```
y_test_pred = svc.predict(X_test)

print("accuracy: {:.2f}".format(accuracy_score(Y_test, y_test_pred)))
print("Precision: {:.2f}".format(precision_score(Y_test, y_test_pred, average = 'micro')))
print("Recall: {:.2f}".format(recall_score(Y_test, y_test_pred, average = 'micro')))
print("F1-score: {:.2f}".format(f1_score(Y_test, y_test_pred, average = 'micro')))
```

```
accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1-score: 1.00
```

Accuracy	1.00
F1_score	1.00
Recall	1.00
Precision	1.00

04. Result

01. 테스트 데이터 전처리

```
# face와 test 각각 배열에 저장
face_test = []
iris_test = []

for filename in order_list:
    imagePath = path + '/04_multimodal_test/' + filename

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

    new_image = (image / 255.).astype(np.float32)
    new_image = cv2.resize(new_image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)

    if filename[-8:-4] == 'face':
      face_test.append(face_descriptor.predict(np.expand_dims(new_image, axis=0))[0])
    elif filename[-8:-4] == 'iris':
      iris_test.append(face_descriptor.predict(np.expand_dims(new_image, axis=0))[0])
```

주어진 테스트 데이터도 전처리하고 모델의 특징점을 생성한다.

```
test_embeddings = []

i = 0
for i in range(len(face_test)):
  test_embeddings.append(face_test[i] + iris_test[i])
  i += 1
```

```
# face model predict
predict = svc.predict(test_embeddings)
```

```
# face model result
predict
```

```
array([27, 31, 22, 57, 54, 45,  5, 12,  8,  7, 36,  2, 46, 50, 10, 61, 49,
       59, 56, 13, 37, 30, 25,  0, 41, 58,  9, 20, 48, 47, 42, 24,  3,  6,
       55, 63, 21, 16, 34, 11, 35, 32, 40, 62, 19, 52, 33, 15,  1, 29, 17,
       26, 53, 28, 18, 14, 43, 60,  4, 39, 38, 23, 51, 44])
```

얼굴과 홍채에서 추출한 테스트 데이터를 합치고

SVM 모델을 이용해 결과를 Predict 한다.

04. Result

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

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

```
result
```

	Image	Answer
0	0.0	27.0
1	1.0	31.0
2	2.0	22.0
3	3.0	57.0
4	4.0	54.0
...	...	...
59	59.0	39.0
60	60.0	38.0
61	61.0	23.0
62	62.0	51.0
63	63.0	44.0

64 rows × 2 columns

최종 결과를 csv 파일로 export한다.

05. Evaluation

05. 평가

- 지난 번 얼굴 데이터를 분류하기 위해 사용한 vgg16 모델로 얼굴과 홍채의 특징점을 추출했다.
- 단일 모델을 이용했을 때보다 각 데이터의 특징점을 추출해 두 가지 모두를 이용해 분류했을 때 더 나은 성능을 볼 수 있었다.
- 이번에 시도한 vgg16 모델이 홍채보다는 얼굴의 특징점을 더 잘 찾아냈는데, 홍채 단일 분류의 정확성도 높은 모델을 찾는다면 더 많은 데이터에서도 확실한 분류가 가능한 모델을 생성할 수 있지 않을까 한다.