# 생체인식보안

얼굴 인식 #2

사이버보안전공 1871095 문수빈

# 01. Training Data

## 01. CNN 모델 형성

트레이닝 데이터에서 특징점 추출을 위해 VGG의 CNN 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 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

## 01. CNN 모델 형성

CNN 모델은 기존 VGG 얼굴 인식 모델을 참고해 구성했다.

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

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

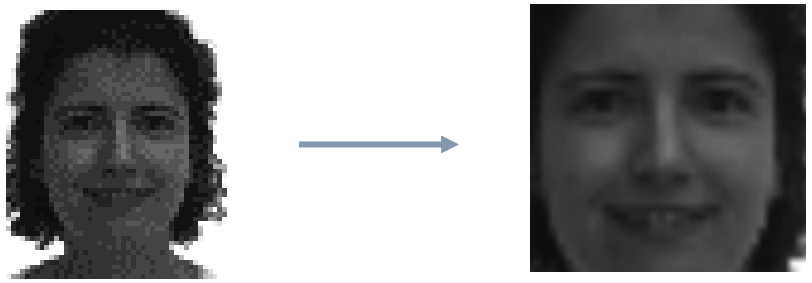
확인 차 트레이닝 데이터셋을 불러와 이미지를 255로 나눠 스케일을 조정하고, 이미지의 크기를 resize 한다.

위에서 생성한 CNN 모델 (face_descriptor) 을 이용해 트레이닝 데이터의 predict를 구하고 이 특징점 결과값들을 embedding_vector에 담는다.

```
tmpPath = "./Face_Training/0124_0002.BMP"
tmp_image = image_load(tmpPath)
tmp_image = (tmp_image / 255.).astype(np.float32)
tmp_image = cv2.resize(tmp_image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)

embedding_vector = face_descriptor.predict(np.expand_dims(tmp_image, axis=0))[0]
```

# 01. Load Image

## 02. 데이터 전처리



- 전체 데이터 영역에서 얼굴 부분만 추출해 학습에 이용하기 위해 데이터 전처리를 한다.
- 데이터의 전처리는 haarcascade_frontalface_default.xml 모듈을 이용하며, 전체 사진에서 얼굴 영역만 파악할 수 있다.

```
def image_load(imagePath):
    global cropped_face
    cascPath = "haarcascade_frontalface_default.xml"
    faceCascade = cv2.CascadeClassifier(cascPath)

    # image = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
    image = cv2.imread(imagePath)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minSize=(15, 15))

    for (x, y, w, h) in faces:
        cropped_face = image[y:y+h, x:x+w]

    return cropped_face
    # cv2.imshow("Face Detected", image)
    # cv2.waitKey(0)
```

1. cv2를 사용해 데이터를 BGR2GRAY로 변환하고, 얼굴의 최소 영역이 될 사이즈를 지정한다.

2. 얼굴 영역으로 추출된 부분만 크롭해 그 결과를 반환한다.

# 01. Training Data

## 03. 특징점 추출

```
# 이미지를 350개의 카테고리로 분류
nb_classes = 350

label= []
embeddings = np.zeros((nb_classes*3, 2622))

i = 0

for filename in train_order:
    category = filename.split(" ")[0]
    imagePath = path + '/02_face_training/' + filename

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

    # face_descriptor 모델에서 나온 특징점들을 저장
    embeddings[i]= face_descriptor.predict(np.expand_dims(image, axis=0))[0]
    label.append(int(category)-1)
    i += 1
```

1

2

3

1. 트레이닝 데이터를 불러온다.
2. image 데이터의 타입과 크기를 변환해 저장한다.
3. face_descriptor 모델을 이용해 얻게 된 특징점 결과 값을 embeddings 배열에 넣고, label 배열에 트레이닝 데이터의 라벨을 저장한다.

```
# 훈련데이터셋 형성
train_data = [embeddings, label]
(X, Y) = (train_data[0], train_data[1])
```

특징점이 담긴 embeddings와 label을 train_data로 지정한다.

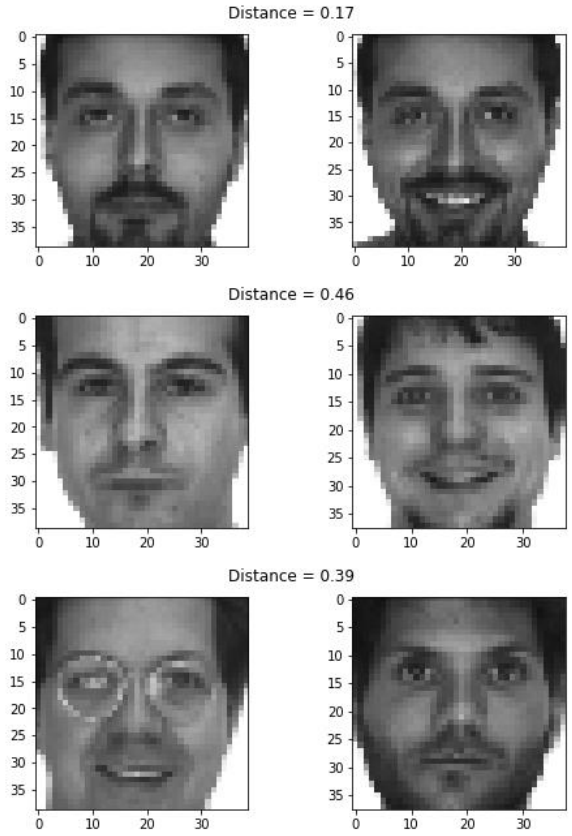
# 01. Training Data

## 03. 특징점 추출

이렇게 특징점들을 저장할 경우, 각 특징점들 간의 distance를 구해 face recognition을 수행할 수 있다.

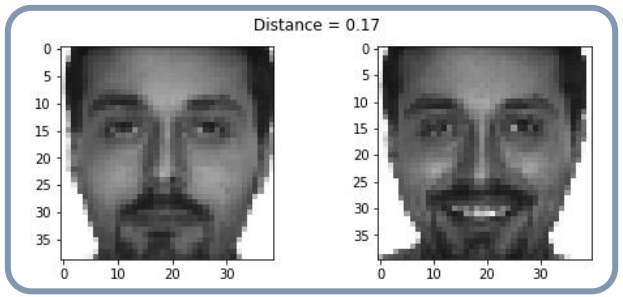
```python
def distance(emb1, emb2):
    return np.sum(np.square(emb1 - emb2))
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8,3))
plt.suptitle(f'Distance = {distance(embeddings[9], embeddings[10]):.2f}')
plt.subplot(121)
plt.imshow(image_load(path +'02_face_training/0004_0001.BMP'))
plt.subplot(122)
plt.imshow(image_load(path +'02_face_training/0004_0002.BMP'))

plt.figure(figsize=(8,3))
plt.suptitle(f'Distance = {distance(embeddings[0], embeddings[5]):.2f}')
plt.subplot(121)
plt.imshow(image_load(path +'02_face_training/0001_0001.BMP'))
plt.subplot(122)
plt.imshow(image_load(path +'02_face_training/0002_0002.BMP'))

plt.figure(figsize=(8,3))
plt.suptitle(f'Distance = {distance(embeddings[8], embeddings[12]):.2f}')
plt.subplot(121)
plt.imshow(image_load(path +'02_face_training/0003_0002.BMP'))
plt.subplot(122)
plt.imshow(image_load(path +'02_face_training/0005_0001.BMP'))
```

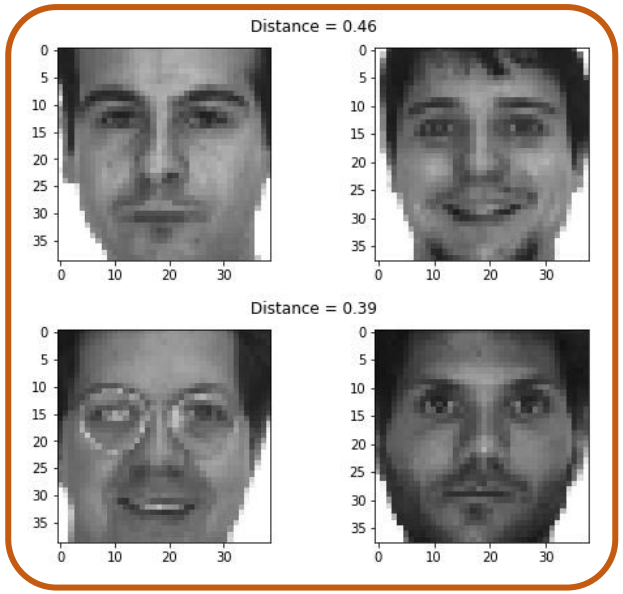


# 01. Training Data

## 03. 특징점 추출




동일인물일 경우 Distance가 0.17이다.




다른 인물일 경우 Distance가 0.46, 0.39로 값이 상대적으로 크다.

```
# 10개의 데이터마다 train과 test 세트로 분리
train_idx = np.arange(nb_classes*3) % 10 != 0
test_idx = np.arange(nb_classes*3) % 10 == 0

X_train = embeddings[train_idx]
X_test = embeddings[test_idx]

targets = np.array(label)

y_train = Y[train_idx]
y_test = Y[test_idx]
```

분류 학습을 위해 10개의 데이터마다 train과 test로 데이터를 분리한다.

# 02. Data Training

## 01. SVM을 이용한 훈련

SVM은 기계학습 분야 중 하나로, 패턴 인식과 자료 분석을 이용한 지도학습 모델이다. 주로 분류와 회귀 분석을 위해 사용된다.

카테고리 중 하나에 속한 데이터의 집합이 주어졌을 때, 데이터 집합을 바탕으로 새로운 데이터의 선형 분류 모델을 생성한다.

```
# 3번의 cross validation을 통해 training set score 교차 검증
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC

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

print("Accuracy on training set : ", cross_val_score(svc, X_train, y_train_label, cv=3))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test_scale, y_test_label)))
```

svc = LinearSVC()로 sklearn의 선형 SVM 모델을 이용하며, 3번의 cross validation을 통해 training set을 교차 검증한다.

```
Accuracy on training set: {:.2f} [0.93650794 0.94920635 0.93333333]
Accuracy on test set: {:.2f} 0.9333333333333333
```

3번의 training set 교차 검증 정확도는 평균 0.933, test set의 정확도는 0.93이 나왔다.

# 02. Data Training

## 02. KNN을 이용한 훈련

KNN 역시 기계학습 분야 중 하나로, 지도학습 모델이다. 주로 분류와 회귀 분석을 위해 사용된다.

이전의 SVM과 동일한 방식으로 sklearn의 KNN 모델을 import해 학습에 사용하였다.

```
# KNN으로 학습 데이터 훈련
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
knn.fit(X_train, y_train_label)

print("Accuracy on training set: {:.2f}".format(knn.score(X_train, y_train_label)))
print("Accuracy on test set: {:.2f}".format(knn.score(X_test, y_test_label)))
```

```
Accuracy on training set: 0.99
Accuracy on test set: 0.90
```

학습 결과 KNN 모델의 training set Accuracy는 0.99이며, test set Accuracy는 0.90이다.

# 03. Result

**최종 스코어**

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

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

| | |
|---|---|
| Accuracy | 0.93 |
| F1_score | 0.93 |
| Recall | 0.93 |
| Precision | 0.93 |

최종 데이터 훈련 결과 모델의 테스트 정확도는 0.93이다.

# 03. Result

## 01. 테스트 데이터 전처리

```
test_embed = np.zeros((700, 2622))

i = 0
for filename in order_list:
    imagePath = path + '/02_face_test/' + filename

    test_image = image_load(imagePath)
    test_image = (test_image / 255.).astype(np.float32)
    test_image = cv2.resize(test_image, (IMAGE_WIDTH, IMAGE_HEIGHT), cv2.INTER_AREA)

    test_embed[i]= face_descriptor.predict(np.expand_dims(test_image, axis=0))[0]
    test_data.append(np.array(test_image))
    i += 1
```

주어진 테스트 데이터도 동일하게 전처리한 후, 각 인물의 특징점을 추출해 저장한다.

# 03. Result

## 02. 예측 결과 출력

```
svc_predict = svc.predict(test_embed)
knn_predict = knn.predict(test_embed)
```

SVM 분류 결과와, KNN 분류 결과를 각각 예측한다.

```
import csv
f = open(path + 'face_result_svc.csv', 'w', newline='')
wr = csv.writer(f)
```

```
wr.writerow(['Image', 'Answer'])
for i in range(len(svc_predict)):
  wr.writerow([i+1, svc_predict[i]+1])
```

```
f.close()
```

```
f = open(path + 'face_result_knn.csv', 'w', newline='')
wr = csv.writer(f)
```

```
wr.writerow(['Image', 'Answer'])
for i in range(len(knn_predict)):
  wr.writerow([i+1, knn_predict[i]+1])
```

```
f.close()
```

이후 모델의 예측 결과를 각각 다른 csv 파일에 export 한다.

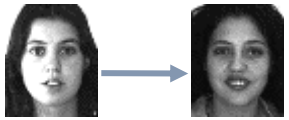
# 03. Result

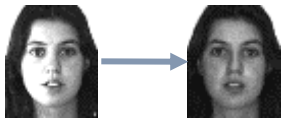
| | A | B |
|---|---|---|
| 1 | Image | Answer |
| 2 | 1 | 5 |
| 3 | 2 | 270 |
| 4 | 3 | 188 |
| 5 | 4 | 219 |
| 6 | 5 | 303 |
| 7 | 6 | 256 |
| 8 | 7 | 306 |
| 9 | 8 | 47 |
| 10 | 9 | 105 |
| 11 | 10 | 153 |
| 12 | 11 | 102 |
| 13 | 12 | 266 |
| 14 | 13 | 216 |
| 15 | 14 | 177 |
| 16 | 15 | 162 |
| 17 | 16 | 131 |
| 18 | 17 | 11 |
| 19 | 18 | 32 |
| 20 | 19 | 151 |
| 21 | 20 | 53 |
| 22 | 21 | 35 |
| 23 | 22 | 249 |
| 24 | 23 | 178 |
| 25 | 24 | 101 |
| 26 | 25 | 189 |
| 27 | 26 | 317 |
| 28 | 27 | 141 |
| 29 | 28 | 141 |
| 30 | 29 | 155 |
| 31 | 30 | 260 |
| 32 | 31 | 175 |
| 33 | 32 | 282 |

KNN

| | A | B |
|---|---|---|
| 1 | Image | Answer |
| 2 | 1 | 5 |
| 3 | 2 | 270 |
| 4 | 3 | 188 |
| 5 | 4 | 219 |
| 6 | 5 | 303 |
| 7 | 6 | 256 |
| 8 | 7 | 306 |
| 9 | 8 | 47 |
| 10 | 9 | 105 |
| 11 | 10 | 153 |
| 12 | 11 | 102 |
| 13 | 12 | 266 |
| 14 | 13 | 216 |
| 15 | 14 | 177 |
| 16 | 15 | 162 |
| 17 | 16 | 82 |
| 18 | 17 | 146 |
| 19 | 18 | 32 |
| 20 | 19 | 151 |
| 21 | 20 | 53 |
| 22 | 21 | 35 |
| 23 | 22 | 249 |
| 24 | 23 | 178 |
| 25 | 24 | 101 |
| 26 | 25 | 189 |
| 27 | 26 | 317 |
| 28 | 27 | 141 |
| 29 | 28 | 141 |
| 30 | 29 | 155 |
| 31 | 30 | 260 |
| 32 | 31 | 175 |
| 33 | 32 | 282 |

SVM

KNN 

SVM 

KNN과 SVM 모두 전반적으로 높은 정확도를 보였지만, 테스트 데이터의 정확도가 SVM이 더 높았으며 실제 예측 결과를 육안으로 비교했을 때 SVM의 분류 결과가 더 정확했다.

모델이 데이터를 분류하지 못했을 경우에는 직전에 분류한 데이터의 결과값이 한 번 더 기록되는 듯 했다. 분류를 못한 얼굴 테스트 데이터의 경우 측면이나 로우 앵글과 같이 학습 데이터와 주어진 각도가 다른 경우인 것을 확인했다.

# 04. Evaluation

## 01. 평가

기존 1차에서 사용한 단순 CNN 모델의 평과 결과 전체 중 132/700의 얼굴 데이터만을 맞추었다.
이를 VGG 모델의 가중치를 이용해 특징점을 학습한 결과, Accuracy가 개선된 것을 확인할 수 있었다.

다만, 모델의 얼굴 영역을 찾아 크롭하고, 그 사진에서 얼굴의 특징점을 찾아냈더니 측면 얼굴의 경우 인식을 잘 하지 못했다.
얼굴에서 특징점을 구한 후 그 정보를 바탕으로 Normalization 하고, 이미지의 중심점을 변환하고 학습을 진행하면 측면 이미지 탐지에서 더 나은 성능을 낼 수 있지 않을까 한다.