

인공지능 Project 1

이름 : 이준휘

학번 : 2018202046

교수 : 박철수 교수님

분반 : 금 3, 4

1. Introduction

해당 과제는 PCA를 이용하여 얼굴을 인식하는 과정을 시뮬레이션을 수행한다. LWF 얼굴 DB를 이용하여 PCA를 통해 eigen-face를 출력하여 보며, KNN 모델을 학습시키는 과정에서 PCA Whitening의 여부를 통한 정확도 차이를 비교한다. 또한 주성분의 개수의 차이를 두어 얼굴 이미지를 재구성하고 결과를 확인한다. KNN 모델은 train과 test 데이터에 PCA를 적용하여 KNN 모델을 생성하고 학습시켜 결과를 설명한다. 또한 해당 결과와 CNN을 이용한 결과와 성능을 비교한다.

2. Algorithm

a. PCA

PCA란 Principal Component Analysis의 약자로 인공지능에서 많이 쓰이는 알고리즘 중 하나다. PCA는 크게 Compression과 Classification으로 핵심을 나눌 수 있다, Compression은 dimensions를 줄인다는 의미로, 기존의 데이터보다 차원의 개수를 줄여 데이터를 본다는 의미다. Classification은 데이터의 패턴을 확인하는 것이다. PCA는 분포가 가장 큰 방향을 탐색하여 탐색한 정보와 가장 관련이 없는 방향부터 탐색을 수행한다. 위의 과정은 Covariance를 행렬곱을 통해 구할 수 있으며 이에 대한 Eigenvalue를 찾는 과정을 통해 고유 벡터를 찾을 수 있다. 해당 과정은 sklearn.decomposition의 PCA 함수를 통해 구현되어 있으며 여기서 component의 개수와 whitening 옵션, random state를 결정여 모델을 fitting, 적용시킬 수 있다. whitening이란 기저 벡터의 값을 eigenvalue 값으로 나누어 정규화하는 기법을 의미한다. 즉 이는 만약 데이터가 multivariable gaussian 분포를 보일 경우 평균이 0이고 공분산이 I(단위 행렬)인 모습을 가진다.

b. KNN

KNN이란 K-nearest neighbors의 약자로 분류, 회귀 문제 등 다양한 문제에 자주 쓰이는 알고리즘이다. 이 때 k가 의미하는 것은 이웃의 숫자를 뜻하며 k만큼의 이웃을 지정하고 그 이웃들의 값을 토대로 계산한다. KNN 분석을 할 때 k를 너무 낮게 설정할 경우 overfitting 문제가 발생하기 때문에 주의해야 하며, k를 너무 높게 줄 경우 데이터가 일반화되기 때문에 유의미한 분석이 어려울 수 있다. k를 선택한 후에는 기준이 되는 데이터와 데이터 간의 유클리드 거리를 계산하고, 가장 가까운 k개의 이웃을 선택하며, 이러한 레코드들의 레이블을 기준으로 분류나 회귀값을 예측한다.

sklearn.neighbors에서 제공하는 KNeighborsClassifier 함수를 이용할 경우 k의 초기값을 설정할 수 있으며 fit() 함수를 통해 최적의 k값을 찾는다. 또한 score 함수를 통해 정확성을 체크할 수 있다.

c. CNN

CNN은 Convolutional Neural Network의 약자로 보통 이미지를 분석하기 위해 패턴을 찾는데 유용한 알고리즘이다. 이미지의 공간정보를 유지하며 학습하며, 필터링 기법을 인공 신경망(NN)에 적용하여 이미지를 더욱 효과적으로 처리한다. CNN이 나오기 이전의 이미지 인식은 2차원으로 이미지를 1차원 배열로 바꾼 후 FNN 신경망을 통해 학습시키는 것이었다. 이는 인접 픽셀 간의 상관관계가 무시된다는 문제가 있었다. 벡터

형태로 표현된 데이터를 입력받기 위해 이미지를 벡터화 해야 하는데 이 때 인접 픽셀 간의 상관관계를 잃어 정보의 손실이 발생하였다. CNN은 이미지의 형태를 보존하도록 행렬 형태의 데이터를 입력 받기 때문에 이러한 정보 손실을 방지할 수 있다. CNN은 여러 계층을 쌓는 구조로 되어있다. Convolution Layer에서는 이미지에 필터링 기법을 적용하며, Pooling layer에서는 이미지의 작은 부분들을 대표 스칼라 값으로 변환시켜 이미지 크기를 줄이는 과정과 같은 다양한 역할을 수행한다.

3. Result

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_lfw_people
import matplotlib
import numpy as np
import pandas as pd

people = fetch_lfw_people(min_faces_per_person = 20, resize = 0.7)
image_shape = people.images[0].shape

import matplotlib.pyplot as plt from sklearn.datasets import fetch_lfw_people
people = fetch_lfw_people(min_faces_per_person=20, resize = 0.7) image_shape = people.image[0].shape


print("people.images.shape : {}".format(people.images.shape))
print("Class : {}".format(people.target_names.size))

people.images.shape : (3023, 87, 65)
Class : 62

dict_keys(['data', 'images', 'target', 'target_names', 'DESCR'])

fig, axes = plt.subplots(2, 5, subplot_kw={'xticks': (), 'yticks': ()})

for target, image, ax in zip(people.target, people.images, axes.ravel()):
    ax.imshow(image)
    ax.set_title(people.target_names[target])
plt.show()
```



위의 그림은 sklearn으로부터 lfw people 데이터를 불러오는 모습이다. 각 인물의 최소 사진 수는 20으로 설정되었으며 resize는 사진의 크기로 0.7배로 조정하였다. 값이 정상적으로 들어갔는지 확인하기 위해 people의 모양과 class의 개수를 출력하였을 때 다음과 같이 나왔다. 해당 image는 87 * 65 크기의 사진이 3023개 존재하며 이름의 종류는 62개가 존재한다. target에는 각 사진에 해당하는 이름 순서가 저장되어 있다. 각 인자에 대한 설명은 DESCR를 통해 확인할 수 있다.

```
counts = np.bincount(people.target)
for i, (count, name) in enumerate(zip(counts, people.target_names)):
    print('{0:30} {1:3}'.format(name, count))
✓ 0.3s
```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

| | |
|-----------------------|-----|
| Alejandro Toledo | 39 |
| Alvaro Uribe | 35 |
| Amelie Mauresmo | 21 |
| Andre Agassi | 36 |
| Angelina Jolie | 20 |
| Ariel Sharon | 77 |
| Arnold Schwarzenegger | 42 |
| Atal Bihari Vajpayee | 24 |
| Bill Clinton | 29 |
| Carlos Menem | 21 |
| Colin Powell | 236 |
| David Beckham | 31 |
| Donald Rumsfeld | 121 |
| George Robertson | 22 |
| George W Bush | 530 |
| Gerhard Schroeder | 109 |

```
min_count = np.argmin(counts)
print(people.target_names[min_count] + " " + str(counts[min_count]))
✓ 0.3s
```

Angelina Jolie 20

위의 사진은 각 이름 class에 해당하는 사진이 몇 개가 있는지 확인하는 과정이다. numpy의 bincount 함수를 통해 사진의 빈도를 알아내고 이를 출력하였으며, 이 때 사진의 개수는 각 인물마다 편차가 있는 것으로 확인된다. 인물의 최소 개수를 알아내기 위해 argmin() 함수를 통해 최소인 counts를 찾아내었고, 해당하는 idx 번호를 탐색하였을 때 20개의 사진을 가진 Angelina Jolie이 나왔다.

```
idx = np.zeros(people.target.shape, np.bool_)
for target in np.unique(people.target):
    idx[np.where(people.target == target)[0][:min_count]] = 1
✓ 0.2s
```

```
x_people = people.data[idx]
y_people = people.target[idx]
✓ 0.3s
```

```
from sklearn.decomposition import PCA
✓ 0.2s
```

PCA에서 인물마다 사진의 개수가 다른 편차를 없애기 위해 가장 작은 사진을 가진 인물을 선택하는 과정이 나온다. idx 배열을 target의 크기만큼 0을 가지도록 설정한 후 반복문을 통해 min_count에 해당하는 배열까지만을 1로 설정한다. 이후 x_people과 y_people 변수를 만들며 이는 각 인물에서 20장만 선택된 idx 배열을 통해 데이터를 옮긴다.

```

pca = PCA(n_components=100, whiten=True, random_state=0)
pca.fit_transform(x_people)
x_pca = pca.transform(x_people)

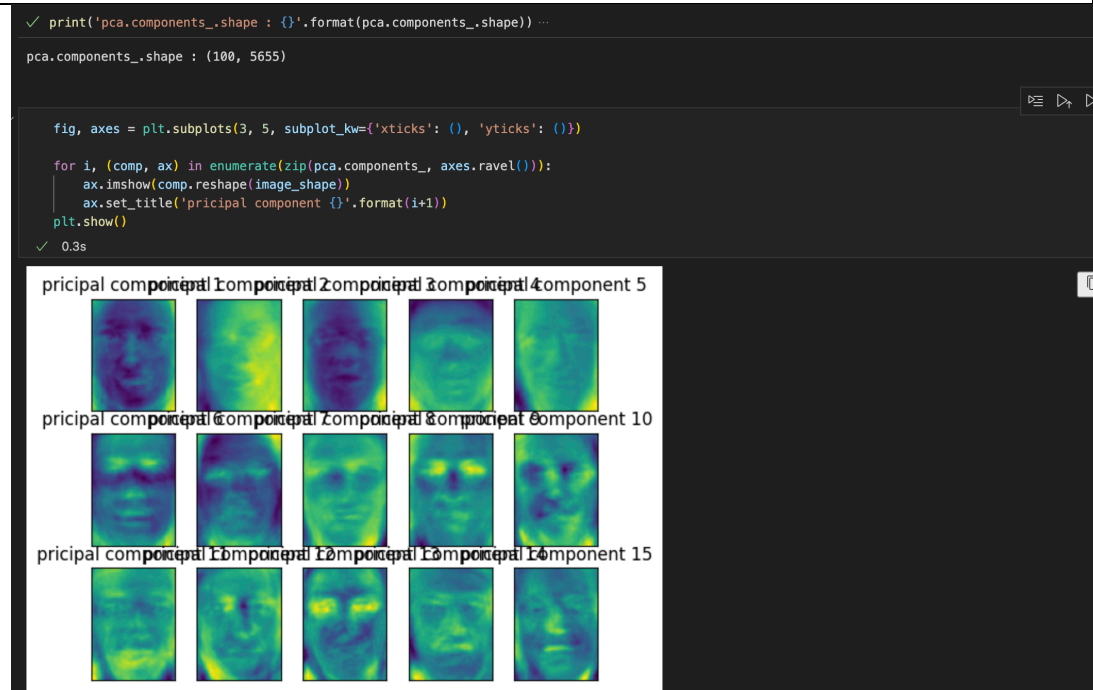
✓ 0.1s

✓ pca.explained_variance_ratio_ ...

array([0.21679473, 0.11694986, 0.08270459, 0.06791736, 0.03418766,
       0.0289731 , 0.02641824, 0.02204424, 0.02042151, 0.0184048 ,
       0.01491712, 0.01407442, 0.01396828, 0.01228028, 0.01128036,
       0.01044049, 0.00942852, 0.00923224, 0.00858377, 0.00808308,
       0.00707099, 0.00689626, 0.00622275, 0.00614205, 0.0058913 ,
       0.00571393, 0.00542919, 0.00493986, 0.00468821, 0.0045548 ,
       0.00435159, 0.00431622, 0.00421781, 0.00395978, 0.00393348,
       0.00382151, 0.0037747 , 0.00355661, 0.00345812, 0.00340897,
       0.00335467, 0.00329736, 0.00318361, 0.00308241, 0.00300633,
       0.00288338, 0.00276826, 0.00271778, 0.00269329, 0.00264369,
       0.00255392, 0.00250668, 0.00238706, 0.00228986, 0.00222468,
       0.0021856 , 0.00213478, 0.00208354, 0.00203003, 0.00197648,
       0.00191465, 0.00188232, 0.00187767, 0.00180287, 0.0017895 ,
       0.00173571, 0.0016941 , 0.0016533 , 0.00162637, 0.00162085,
       0.00158118, 0.00153627, 0.00148961, 0.00144968, 0.00143395,

```

PCA 함수를 통해 component의 개수를 100으로 제한하며 whitening 옵션을 켜 상태로 PCA를 수행하였을 때 다음과 같은 결과가 나왔다. pca에서 측정된 분산은 다음과 같이 크기 순으로 정렬되어 있다.



다음 사진은 각 component를 출력하는 모습이다. 다음과 같이 얼굴의 고유 성분이 추출되어 나온 것을 확인할 수 있다. 이를 통해 PCA가 정상적으로 구현되었음을 알 수 있다.

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics

✓ 0.3s

scaler = MinMaxScaler()
x_people_scaled = scaler.fit_transform(x_people)

x_train, x_test, y_train, y_test = train_test_split(x_people_scaled, y_people, stratify=y_people, random_state=0)

✓ 0.1s

```

해당 사진은 기존 사진 사람 데이터를 늘려서 train과 test로 구분하기 위해 sklearn의

MinMaxScaler를 이용한다. 이를 통해 `x_people`을 늘린 데이터를 사용한다. `train_test_split` 함수는 `test`와 `train`으로 데이터를 나누어주는 함수다. 이를 통해 `x_train`, `x_test`, `y_train`, `y_test`로 데이터를 나누어준다.

```
(parameter) X_test: Any
def pca_faces(X_train, X_test):
    reduced_images = []
    for n_components in [10, 25, 50, 100]:
        pca = PCA(n_components=n_components)
        pca.fit(X_train)
        X_test_pca = pca.transform(X_test)
        X_test_back = pca.inverse_transform(X_test_pca)
        reduced_images.append(X_test_back)
    return reduced_images
```

✓ 0.6s

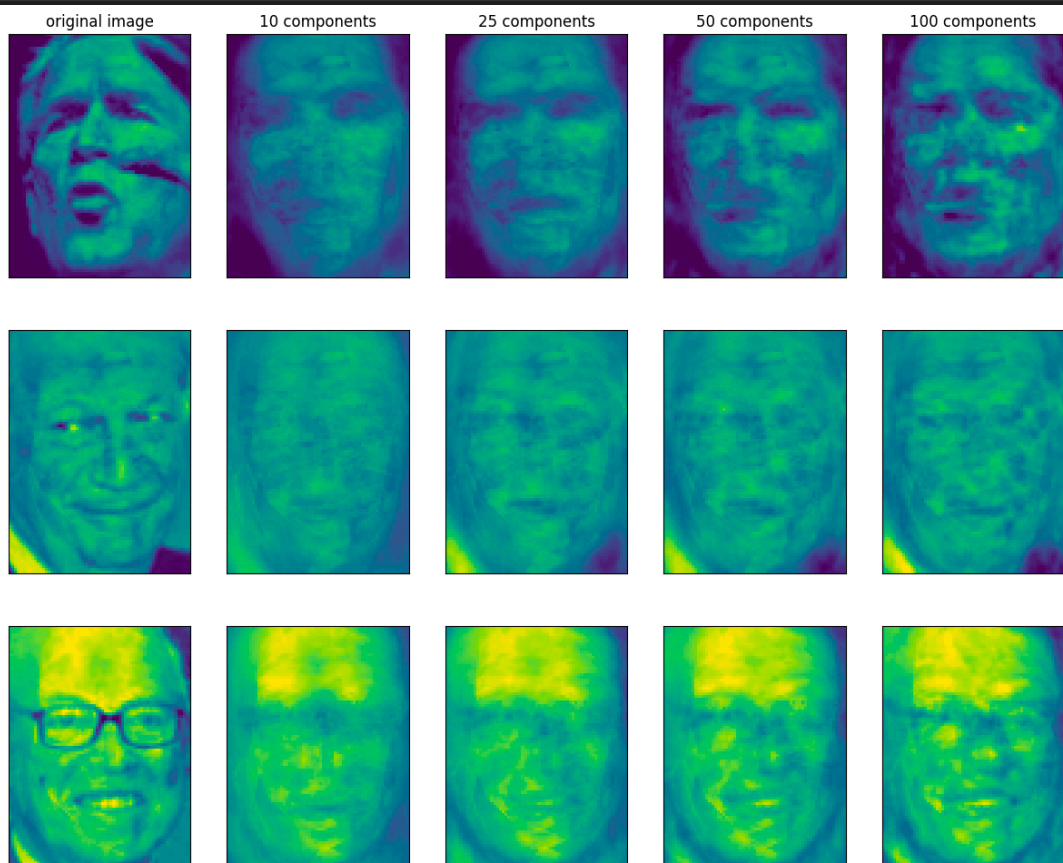
```
def plot_pca_faces(X_train, X_test, image_shape):
    reduced_images = pca_faces(X_train, X_test)
    fig, axes = plt.subplots(3, 5, figsize=(15, 12),
                             subplot_kw={'xticks': (), 'yticks': ()})
    for i, ax in enumerate(axes):
        ax[0].imshow(X_test[i].reshape(image_shape),
                     vmin=0, vmax=1)
        for a, X_test_back in zip(ax[1:], reduced_images):
            a.imshow(X_test_back[i].reshape(image_shape), vmin=0, vmax=1)

    axes[0, 0].set_title("original image")
    for ax, n_components in zip(axes[0, 1:], [10, 25, 50, 100]):
        ax.set_title("%d components" % n_components)
```

✓ 0.4s

```
plot_pca_faces(x_train, x_test, image_shape)
```

✓ 0.8s



위의 사진은 component의 개수의 따라 이미지를 재구성하였을 때의 차이를 보는 함수를 만든 것이다. 위의 사진을 보았을 때 component의 수가 많아질 수록 이미지를 재구성하였을 때 더욱 원본에 가까워지는 것을 확인할 수 있다.

```

pca = PCA(n_components=100, whiten=False, random_state=0)
pca.fit(x_train)
x_train_pca = pca.transform(x_train)
x_test_pca = pca.transform(x_test)
✓ 0.2s

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train_pca, y_train)
✓ 0.3s

KNeighborsClassifier(n_neighbors=1)

print('knn.score(x_test, y_test) : {:.3f}'.format(knn.score(x_test_pca, y_test)))
✓ 0.4s

knn.score(x_test, y_test) : 0.065

```

해당 사진은 whitening 옵션을 적용하지 않을 때의 KNN 모델을 확인하는 것이다. KNN 모델은 KneighborClassifier() 함수를 통해 불러올 수 있으며 fit를 통해 해당 데이터를 학습시킬 수 있다. whitening이 적용되지 않은 pca를 사용한 knn의 정확도는 약 0.065로 상당히 낮게 나온 모습이다.

```

pca = PCA(n_components=100, whiten=True, random_state=0)
pca.fit(x_train)
x_train_pca = pca.transform(x_train)
x_test_pca = pca.transform(x_test)
✓ 0.1s

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train_pca, y_train)
✓ 0.5s

KNeighborsClassifier(n_neighbors=1)

✓ print('knn.score(x_test, y_test) : {:.3f}'.format(knn.score(x_test_pca, y_test))) ...
knn.score(x_test, y_test) : 0.081

```

해당 사진은 whitening 옵션을 적용하였을 때의 KNN 모델을 확인하는 것이다. whitening이 적용된 knn 모델의 정확도는 0.082이 나오는 모습을 확인할 수 있다. 이를 통해 whitening을 적용하였을 때 정확도가 상승한다는 것을 알 수 있다.

```

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
✓ 0.2s

from keras.utils import to_categorical
from keras.applications.resnet50 import preprocess_input
from sklearn.model_selection import train_test_split
✓ 0.2s

X_people = people.images
Y_people = people.target

X_people = X_people.reshape(X_people.shape[0], image_shape[0], image_shape[1], 1)
X_people.shape

face_labels = to_categorical(Y_people)

X_train, X_test, Y_train, Y_test = train_test_split(X_people, face_labels, stratify=face_labels, random_state=0)
✓ 0.2s

```

다음 그림은 CNN 모델을 학습하는 과정을 보이는 코드다. X_people과 Y_people은 전체 데이터를 사용할 예정이며, X_people을 CNN에 넣기 위해서는 형태를 이에 맞게 4차원으로 변경시킬 필요가 있다. 사진의 개수, 사진의 크기, gray_scale의 이미지임으로 마지막을 1로 설정한 형태로 모양을 변경시킨다. face_labels 변수는 Y_people을 one-hot encoding을 통해 형태를 변형한 모습을 갖는다. one-hot encoding은 keras에서 제공하는 to_categorical 함수를 이용한다. 이 데이터를 이전에 사용한 train_test_split 함수를 통해 train과 test로 분리한다.

```

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_shape[0], image_shape[1], 1)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(people.target_names.size, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

```

✓ 0.3s

Model: "sequential_36"

| Layer (type) | Output Shape | Param # |
|-------------------------------|--------------------|---------|
| ===== | | |
| conv2d_87 (Conv2D) | (None, 85, 63, 32) | 320 |
| ===== | | |
| max_pooling2d_85 (MaxPooling) | (None, 42, 31, 32) | 0 |
| ===== | | |
| conv2d_88 (Conv2D) | (None, 40, 29, 64) | 18496 |
| ===== | | |
| max_pooling2d_86 (MaxPooling) | (None, 20, 14, 64) | 0 |
| ===== | | |
| conv2d_89 (Conv2D) | (None, 18, 12, 64) | 36928 |
| ===== | | |
| max_pooling2d_87 (MaxPooling) | (None, 9, 6, 64) | 0 |
| ===== | | |
| flatten_36 (Flatten) | (None, 3456) | 0 |
| ===== | | |
| dense_71 (Dense) | (None, 1024) | 3539968 |
| ===== | | |
| dense_72 (Dense) | (None, 62) | 63550 |
| ===== | | |

위의 사진은 임의의 CNN layer를 적층하는 모습이다. CNN 모델의 사용은 Sequential 함수를 통해 할 수 있으며 add()함수를 통해 적층할 수 있다. Conv2D layer은 특정 filter size로 matrix convolution을 수행하며, MaxPooling을 통해 값을 조정한다. Flatten은 1차원 데이터로 조정시키는 역할을 수행하며, Dense를 통해 특정 개수로 출력을 모아준다. 해당 모델이 다음과 같이 정상적으로 생성되었음을 보았다.

```

model.fit(X_train, Y_train, epochs=20, batch_size=32, verbose=1)

```

✓ 11m 8.3s

출력이 축소됨 ...

```

score = model.evaluate(X_test, Y_test, verbose=0)

```

✓ 5.9s

```

print("Test loss = {}".format(score[0]))
print("Test accuracy = {}".format(score[1]))

```

✓ 0.3s

Test loss = 4.609077842147262

Test accuracy = 0.5052909851074219

Model을 fit함수를 이용해 epoch, batch size를 설정하여 학습시킨다. epoch는 학습 횟수를 나타내며, batch size는 트레이닝 그룹을 작게 나누었을 때 하나의 소그룹에 속하는 데이터 수다. 모델을 test 데이터를 통해 평가하였을 때 loss는 4.6, accuracy는 0.5가 나온다. 이전의 pca를 적용한 KNN 모델(0.08)보다 정확도가 훨씬 높게 상승하였다.

4. Consideration

해당 과제를 통해 PCA, KNN, CNN을 기존 라이브러리를 통해 손쉽게 구현할 수 있음을 알 수 있다. PCA의 whitening 옵션을 사용하는 것이 정확도가 상승한다는 사실을 알 수 있다. 또한 PCA를 사용한 KNN의 모델은 정확도가 0.08이 나오고 CNN 모델은 정확도가 0.5가 나오기에 두 모델 중 하나의 모델을 선택한다면 CNN 모델을 사용하는 것이 맞다는 결론을 낼 수 있다. 과제를 통해 이러한 다양한 사실을 직접 실습을 통해 체험할 수 있는 과제였다.

