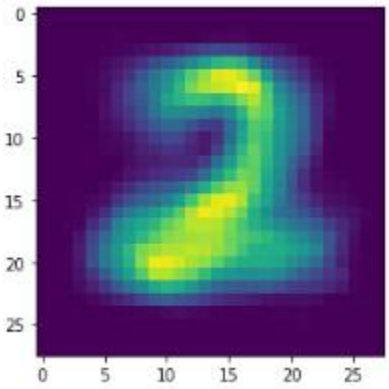
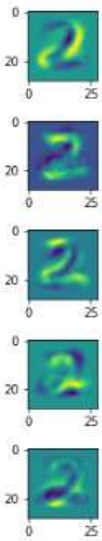
< CSE 302 Assignment 4 >

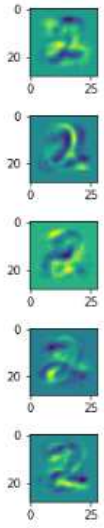
202011212 진호진

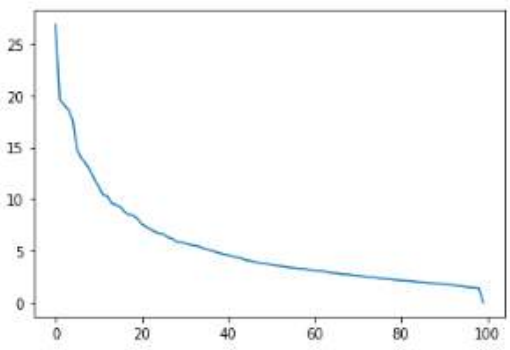
1)

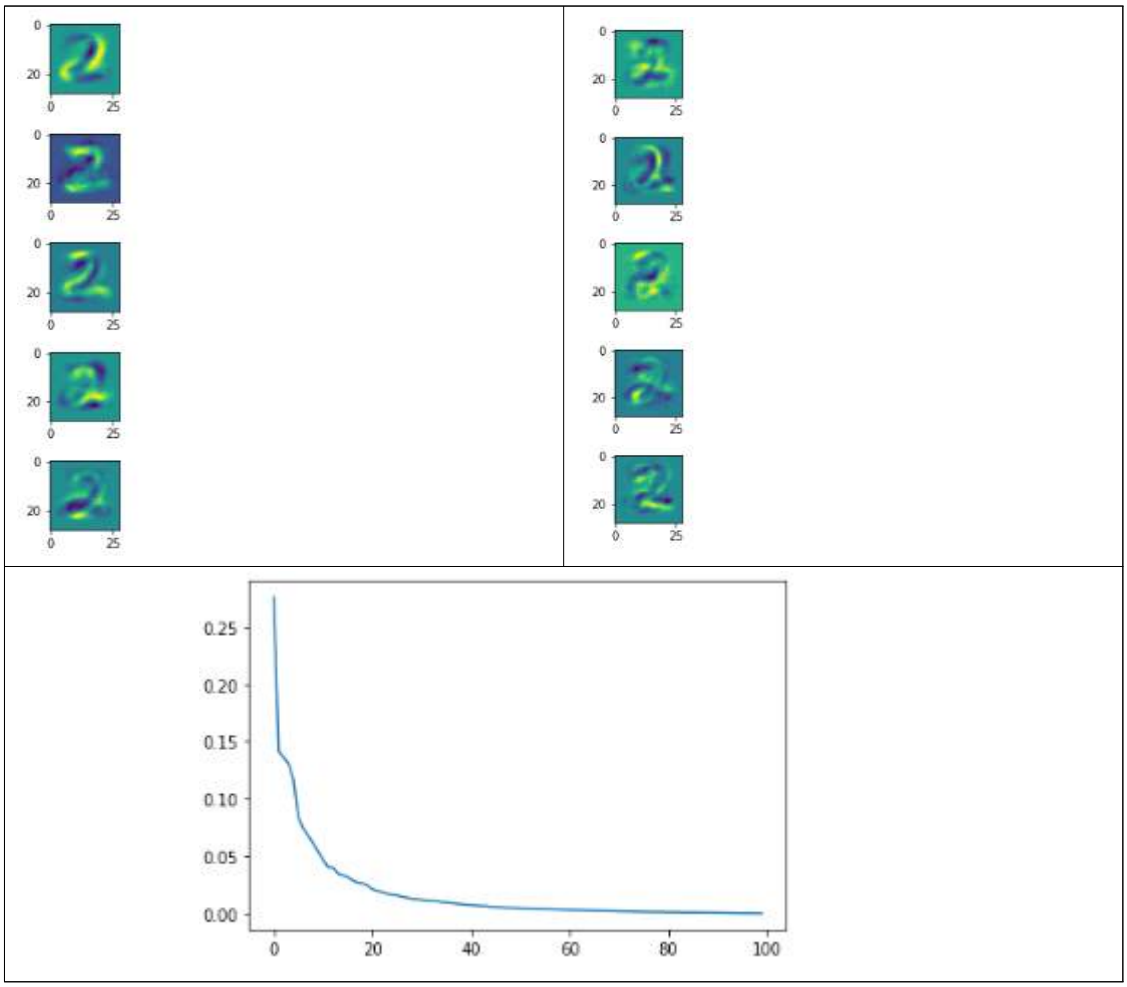
```
mean = np.asarray(img).mean(axis = 0)
plt.imshow(mean.reshape(28, 28))
plt.show()

pca = PCA(n_components = 100)
pca.fit_transform(img)


for eigenvector in pca.components_[0:10]:
  plt.subplot(2, 5, 1)
  plt.imshow(eigenvector.reshape(28, 28))
  plt.show()


eigenvalue = pca.singular_values_
plt.plot(range(0, 100), eigenvalue)
plt.show()
```



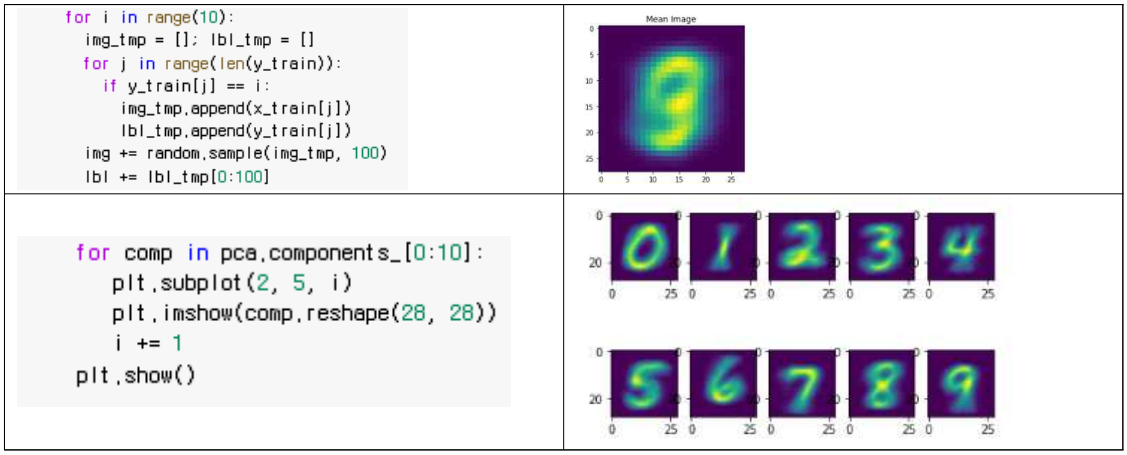







PCA 와 Kernel PCA의 코드가 거의 똑같기 때문에 PCA 코드만 위의 표에 첨부하였다. Sigmoid로 작동시킨 Kernel PCA는 일반 PCA와 거의 유사한 것을 알 수 있었다. 그래프 또한 개형이 매우 비슷하게 출력되어 이를 더 확실하게 알 수 있었다.

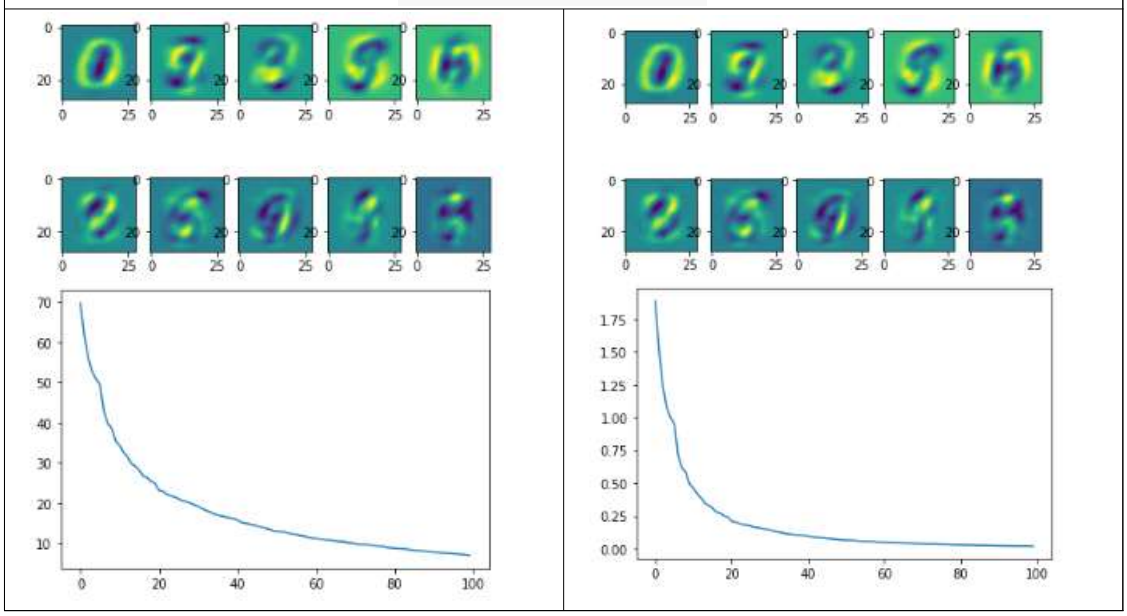
2)




먼저 평균 이미지와 10개를 출력한 것이다. 평균 이미지는 숫자가 섞여있는 느낌을 받을 수

있다.

```
pca = PCA(n_components=100)
pca.fit_transform(img)

i = 1
for comp in pca.components_[0:10]:
    plt.subplot(2, 5, i)
    plt.imshow(comp.reshape(28, 28))
    i += 1
plt.show()

eigval = pca.singular_values_
plt.plot(range(0, 100), eigval)
plt.show()
```



클래스에서 무작위로 추출하여 결과를 도출하여 숫자가 선명하지 않고 섞여 있는 느낌을 그대로 받을 수 있다. eigenvalue 값에도 첫번째 문제와 조금 다른 점이 나타났는데, 이는 다른 데이터를 구분할때 PCA를 사용해도 좋다는 결론으로 받아들였다.

3)

```
from sklearn.metrics.cluster import rand_score
from sklearn.metrics.cluster import mutual_info_score

pca = PCA(n_components = 100)
img_pca = pca.fit_transform(img)
Kmeans = KMeans(n_clusters = 10)
Kmeans.fit(img_pca)

prediction = Kmeans.fit_predict(img_pca)

print("rand_index:",rand_score(lbl, prediction))
print("mutual_info_score:", mutual_info_score(lbl, prediction))
```

```
rand_index: 0.8751391391391391
mutual_info_score: 1.1199616092442761
```

```
from sklearn.metrics.cluster import rand_score
from sklearn.metrics.cluster import mutual_info_score

pca = KernelPCA(n_components = 100, kernel="sigmoid", gamma=0.001)
img_pca = pca.fit_transform(img)
Kmeans = KMeans(n_clusters = 10)
Kmeans.fit(img_pca)

prediction = Kmeans.fit_predict(img_pca)

print("rand_index:",rand_score(lbl, prediction))
print("mutual_info_score:", mutual_info_score(lbl, prediction))
```

```
rand_index: 0.8805985985985986
mutual_info_score: 1.113049623471206
```

Rand_index는 무작위 clustering에서도 높은 값이 나올 확률이 있지만 그 값이 현재 0.88로 1과 매우 가깝게 나온 것으로 봐서는 성능이 꽤 좋다는 것을 알 수 있다. mutual_info_score이 높게 나온것으로는 분포의 유사도가 떨어진다는 것을 알 수 있다.

4)

```
from sklearn.metrics import pairwise_distances_argmin_min
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

pca = PCA(n_components = 100)
img_pca = pca.fit_transform(img)
Kmeans = KMeans(n_clusters = 10)
Kmeans.fit(img_pca)

centers,_ = pairwise_distances_argmin_min(Kmeans.cluster_centers_, img_pca)

Xt=[]
Yt=[]

for i in range(0,10):
  Xt.append(img[centers[i]])
  Yt.append(lbl[centers[i]])

KNN=KNeighborsClassifier(n_neighbors = 1) #1-NN
KNN.fit(Xt,Yt)
prediction_knn = KNN.predict(x_test)
print("Accuracy:₩n",classification_report(y_test,prediction_knn))
```

Accuracy:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.87 | 0.74 | 0.80 | 980 |
| 1 | 0.42 | 1.00 | 0.59 | 1135 |
| 2 | 0.79 | 0.46 | 0.58 | 1032 |
| 3 | 0.78 | 0.49 | 0.60 | 1010 |
| 4 | 0.00 | 0.00 | 0.00 | 982 |
| 5 | 0.38 | 0.67 | 0.49 | 892 |
| 6 | 0.81 | 0.77 | 0.79 | 958 |
| 7 | 0.55 | 0.57 | 0.56 | 1028 |
| 8 | 0.00 | 0.00 | 0.00 | 974 |
| 9 | 0.37 | 0.61 | 0.46 | 1009 |
| accuracy | | | 0.54 | 10000 |
| macro avg | 0.50 | 0.53 | 0.49 | 10000 |
| weighted avg | 0.50 | 0.54 | 0.49 | 10000 |

```
from sklearn.metrics import pairwise_distances_argmin_min
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

pca = KernelPCA(n_components = 100, kernel = "sigmoid", gamma = 0.001)
img_pca = pca.fit_transform(img)
Kmeans = KMeans(n_clusters = 10)
Kmeans.fit(img_pca)

centers,_ = pairwise_distances_argmin_min(Kmeans.cluster_centers_, img_pca)

Xt=[]
Yt=[]

for i in range(0,10):
  Xt.append(img[centers[i]])
  Yt.append(lbl[centers[i]])

KNN=KNeighborsClassifier(n_neighbors = 1) #1-NN
KNN.fit(Xt,Yt)
prediction_knn = KNN.predict(x_test)
print("Accuracy:₩n",classification_report(y_test,prediction_knn))
```

Accuracy:

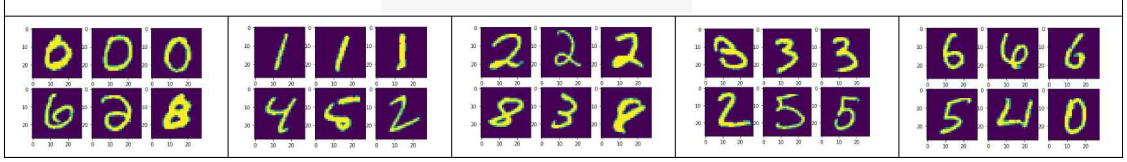
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.86 | 0.75 | 0.80 | 980 |
| 1 | 0.46 | 0.99 | 0.63 | 1135 |
| 2 | 0.82 | 0.43 | 0.56 | 1032 |
| 3 | 0.38 | 0.76 | 0.51 | 1010 |
| 4 | 0.00 | 0.00 | 0.00 | 982 |
| 5 | 0.00 | 0.00 | 0.00 | 892 |
| 6 | 0.57 | 0.87 | 0.69 | 958 |
| 7 | 0.55 | 0.59 | 0.57 | 1028 |
| 8 | 0.00 | 0.00 | 0.00 | 974 |
| 9 | 0.39 | 0.61 | 0.48 | 1009 |
| accuracy | | | 0.51 | 10000 |
| macro avg | 0.40 | 0.50 | 0.42 | 10000 |
| weighted avg | 0.41 | 0.51 | 0.43 | 10000 |

과제 3에서 했던 것 처럼 center를 구해서 1NN을 적용하여 accuracy를 구해보았다. 하지만 너무나도 낮은 정확도인 54퍼센트와 51퍼센트가 결과 값으로 등장하였다. 이를 알아보기 위해서 여러 정보들을 찾아보았지만, 정확하게 알 수 없었고 결국 5번 문제에서 그 답을 찾을 수 있었다.

5)

```
predict = KNN.predict(x_test)
correct = [[] for i in range(10)]
incorrect = [[] for i in range(10)]

for i in range(10):
  for j in range(len(y_test)):
    if y_test[j] != i:
      if predict[j] == i:
        if len(incorrect[i]) < 3:
          incorrect[i].append(j)
    else:
      if predict[j] == i:
        if len(correct[i]) < 3:
          correct[i].append(j)
```

```
[14] i=0
     j=1

     for n in correct[i]:
       img = x_test[n].reshape(28, 28)
       plt.subplot(2, 3, j)
       plt.imshow(img)
       j += 1

     for n in incorrect[i]:
       img = x_test[n].reshape(28, 28)
       plt.subplot(2, 3, j)
       plt.imshow(img)
       j += 1
```



10개를 각각 출력해줬음에도 불구하고 절반정도는 출력이 되지 않았다. 이 값은 매번 실행에 따라 변경되었다. 처음에는 실수로 어떤 class들이 이 과정에서 누락된 줄 알고 모든 값을 찍어보았다. 그랬더니 출력되지 않은 class들은 예측에 성공한 숫자가 단 하나도 없다는 것을 알아냈다. 성공한 case가 없어서 분류가 되지 않았던 것이다. 반면 출력이 되는 숫자들은 정확도가 꽤 높았다. 이 현상의 원인을 자세하게는 알 수 없으나, 중앙값을 구하는 기작이 적용되면서 이러한 결과가 나왔다가 조심스럽게 예측해 볼 수는 있을 것 같다.