Cairo University

Faculty of Engineering
Cairo University

DSP Assignment 3

Submitted to: Dr. Mohsen Rashwan

Name	*SN*	*BN*
Abdelrahman Mostafa Mouse Mohamed Rabah	**3**	**12**
Abdallah Adel Ibrahim	**3**	**15**
Amr Mohamed	**3**	**30**

Data Downloading:

```
import opendatasets

opendatasets.download('https://www.kaggle.com/datasets/mohamedgamal07/reduced-mnist')
```

✓ 24.9s

Making sure Data is correct and divided equally:

```
classes = os.listdir(train_path)
num_classes = len(classes)
source_path=[train_path +f'{a}' for a in classes]
classes_dir=[f'{a}_dir' for a in classes]
for cl_dir,cl_path in zip(classes_dir,source_path):
    print(cl_dir,': ',len(os.listdir(cl_path)))
```

✓ 0.1s

```
0_dir :  1000
1_dir :  1000
2_dir :  1000
3_dir :  1000
4_dir :  1000
5_dir :  1000
6_dir :  1000
7_dir :  1000
8_dir :  1000
9_dir :  1000
```

```python
classes = os.listdir(test_path)
num_classes = len(classes)
source_path=[test_path +f'{a}' for a in classes]
classes_dir=[f'{a}_dir' for a in classes]
for cl_dir,cl_path in zip(classes_dir,source_path):
    print(cl_dir,': ',len(os.listdir(cl_path)))
```

0.2s

```
0_dir :  200
1_dir :  200
2_dir :  200
3_dir :  200
4_dir :  200
5_dir :  200
6_dir :  200
7_dir :  200
8_dir :  200
9_dir :  200
```

Copy data from downloaded folders to arrays:

```python
def copy(data_path):
    imgs = []
    labels = []
    folders = os.listdir(data_path)
    for folder in folders:
        fullpath = os.path.join(data_path, folder)
        images = glob.glob(os.path.join(fullpath,'*.jpg'))
        for img in images:
            labels.append(img.split('/')[-1][0])
            img = cv2.imread(img)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            imgs.append(img)

    return np.array(imgs), np.array(labels)
```

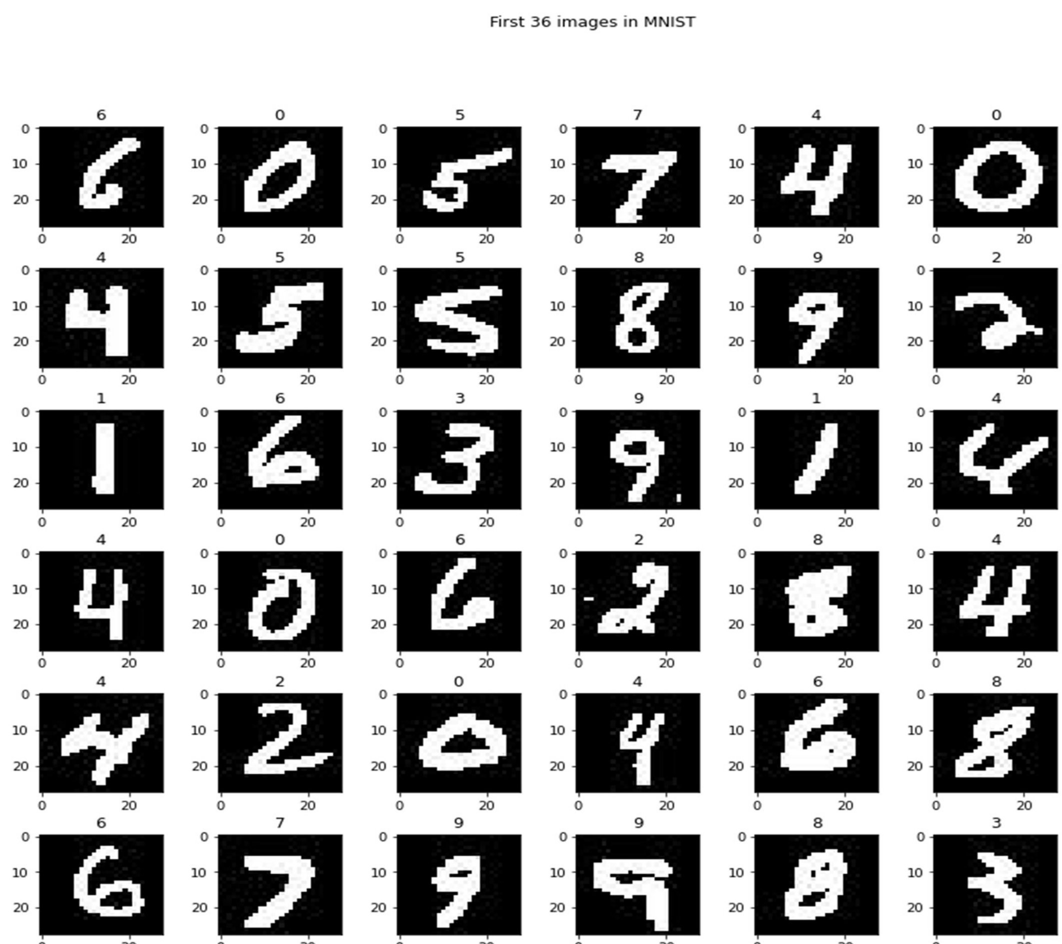
0.2s

Shuffling and Normalizing the data:

```
X_train, y_train = copy(train_path)
X_train = X_train/255.0
X_train, y_train = shuffle(X_train, y_train, random_state=42)
X_test, y_test = copy(test_path)
X_test = X_test/255.0
X_test, y_test = shuffle(X_test, y_test, random_state=42)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
```
✓ 1m 18.2s

Plotting few samples:

```
fig, ax = plt.subplots(6, 6, figsize = (12, 12))
fig.suptitle('First 36 images in MNIST')
fig.tight_layout(pad = 0.3, rect = [0, 0, 0.9, 0.9])
for x, y in [(i, j) for i in range(6) for j in range(6)]:
    ax[x, y].imshow(X_train[x + y * 6].reshape((28, 28)), cmap = 'gray')
    ax[x, y].set_title(y_train[x + y * 6])
```
✓ 5.3s



Reshaping the data to make it 1D instead of 2D (Gray Channels):

```
def unroll(x):
    x = x.reshape(x.shape[0], x.shape[1]*x.shape[2])
    return x
```

The DCT and Zig-Zag method:

```
def dct2(block):
    l = []
    for b in block:
        l.append(dct(dct(b.T, norm='ortho').T, norm='ortho'))
    return l

def small_zigzag(x):
    l = []
    for a in x:
        l.append(np.concatenate([np.diagonal(a[::-1,:], i)[::(2*(i % 2)-1)] for i in range(1-a.shape[0], a.shape[0])])[0:200])
    return np.array(l)
```

DCT Generation:

```
X_tr_dct, t1 = dct2(X_train)
X_tr_dct_zz, t2 = small_zigzag(X_tr_dct)
X_train
print('Generated Train DCT with Zig-Zag in {} seconds, shape of train{}'.format(t1+t2, X_tr_dct_zz.shape))
X_te_dct, t1 = dct2(X_test)
X_te_dct_zz, t2 = small_zigzag(X_te_dct)
print('Generated Test DCT with Zig-Zag in {} seconds, shape of test{}'.format((t1+t2), X_te_dct_zz.shape))
```

✓ 2.8s

```
Generated Train DCT with Zig-Zag in 2.259641170501709 seconds, shape of train(10000, 200)
Generated Test DCT with Zig-Zag in 0.4289977550506592 seconds, shape of test(2000, 200)
```

PCA:

```
pca_n = 0.9
def pca(x_train, x_test, n):
    start = time.time()
    pca = PCA(n_components=n, random_state=42)
    x_train = pca.fit_transform(x_train)
    x_test = pca.transform(x_test)
    end = time.time()
    print(f'time it took PCA:{end - start}, shape of X train: {x_train.shape}, x_test = {x_test.shape}' )
    return x_train, x_test
```

pca_n is variable which indicates to keep 90% of variance

PCA Generation:

```
X_tr_pca = unroll(X_train)
X_te_pca = unroll(X_test)
X_tr_pca, X_te_pca = pca(X_tr_pca, X_te_pca, pca_n)
```

✓ 1.1s

```
time it took PCA:1.0498216152191162, shape of X train: (10000, 167), x_test = (2000, 167)
```

Encoder:

```
import tensorflow as tf
from tensorflow import keras

tf.random.set_seed(42)
np.random.seed(42)

stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),
])
stacked_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])

stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
stacked_ae.compile(loss="binary_crossentropy",
                   optimizer='adam')
history = stacked_ae.fit(X_train, X_train, epochs=15,
                         validation_data=(X_test, X_test))
```

✓ 28.3s

Results:

```
Epoch 14/15
313/313 [==============================] - 2s 6ms/step - loss: 0.1161 - val_loss: 0.1195
Epoch 15/15
313/313 [==============================] - 2s 6ms/step - loss: 0.1154 - val_loss: 0.1184
```

Generation:

```python
X_train_encoded = stacked_ae.predict(X_train)
X_train_encoded_unrolled = unroll(X_train_encoded)
X_test_encoded = stacked_ae.predict(X_test)
X_test_encoded_unrolled = unroll(X_test_encoded)
```

✓ 0.7s

```python
print(X_train_encoded_unrolled.shape, X_test_encoded_unrolled.shape)
```

✓ 0.2s

```
(10000, 784) (2000, 784)
```

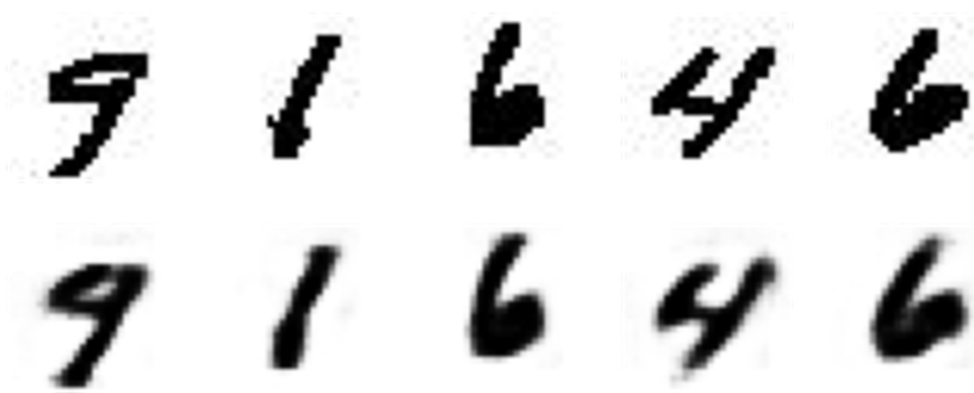
Plotting some generated images:

```python
def plot_image(image):
    plt.imshow(image, cmap="binary")
    plt.axis("off")

def show_reconstructions(model, images=X_test, n_images=5):
    reconstructions = model.predict(images[:n_images])
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(images[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])

show_reconstructions(stacked_ae)
```

✓ 1.4s



KMeans:

```
clusters = [1, 4, 16, 32]
def kmeans(x_train, n):
    start = time.time()
    KMS = KMeans(n_clusters=n, random_state=42)
    KMS.fit(x_train)
    end = time.time()
    return KMS, end - start
def retrieve_info_kmeans(cluster_labels, y_train, model):
    reference_labels = {}
    for i in range(len(np.unique(model.labels_))):
        index = np.where(model.labels_ == i,1,0)
        num = np.bincount(y_train[index==1]).argmax()
        reference_labels[i] = num
    return reference_labels
```

Gaussian Mixtures:

```
mixtures = [1, 2, 4]
def gmm(x_train, n):
    start = time.time()
    GMM = GaussianMixture(n_components=n, random_state=42)
    GMM.fit(x_train)
    end = time.time()
    return GMM, end - start

def retrieve_info_gmm(x, y_train, model):
    labels = model.predict(x)
    reference_labels = {}
    for i in range(len(np.unique(labels))):
        index = np.where(labels == i,1,0)
        num = np.bincount(y_train[index==1]).argmax()
        reference_labels[i] = num
    return reference_labels, labels
```

Support Vector Machines Classifier:

```python
kinds = ['linear', 'rbf']
def svm(x_train, x_test, kind):
    start = time.time()
    SVM = SVC(kernel=kind, random_state=42)
    SVM.fit(x_train, y_train)
    y_hat = SVM.predict(x_train)
    y_pred = SVM.predict(x_test)
    end = time.time()
```

Features												
Classifier	DCT				PCA				Autoencoder			
	Train		Test		Train		Test		Train		Test	
	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time
Kmeans												
Kmeans-1	10.0%	2.777	10.0%	2.556	10.0%	2.703	10.0%	2.511	10.0%	3.644	10.0%	2.736
Kmeans-4	36.52%	1.630	38.3%	0.467	36.51%	1.450	38.35%	0.500	36.01%	7.261	38.25%	1.947
Kmeans-16	72.76%	5.002	77.149%	1.132	73.34%	5.123	76.1%	1.011	73.78%	28.83	77.60%	5.47
Kmeans-32	80.63%	9.391	86.1%	1.652	82.06%	9.507	86.55%	2.328	80.88%	47.386	85.1%	9.656
GMM												
GMM-1	10.0%	0.974	10.0%	0.705	10.0%	1.07	10.0%	0.449	10.0%	4.706	10.0%	1.2409
GMM-2	19.16%	9.217	19.8%	1.021	18.98%	24.626	19.90%	0.889	19.58%	88.09	19.90%	4.558
GMM-4	30.83%	41.42	38.45%	1.423	35.14%	67.23	38.55%	1.448	29.18%	233.52	37.7%	6.1389

SVM	Accuracy Train	Accuracy Test	Total Time	Accuracy Train	Accuracy Test	Total Time	Accuracy Train	Accuracy Test	Total Time
SVM-Linear	98.07%	94.35%	9.264	98.8%	93.8%	5.716	99.039%	94.69%	22.00
SVM-rbf	97.66%	97.35%	17.31	99.19%	97.7%	21.12	97.09%	96.35%	49.86

Time is in seconds. Processing speed depends on CPU/GPU speed. Autoencoder takes so much time since it just generates new noisy images with all features.

an RBF (radial basis function) kernel is used when the boundaries are hypothesized to be curve-shaped.

RBF kernel uses two main parameters, gamma and C that are related to:

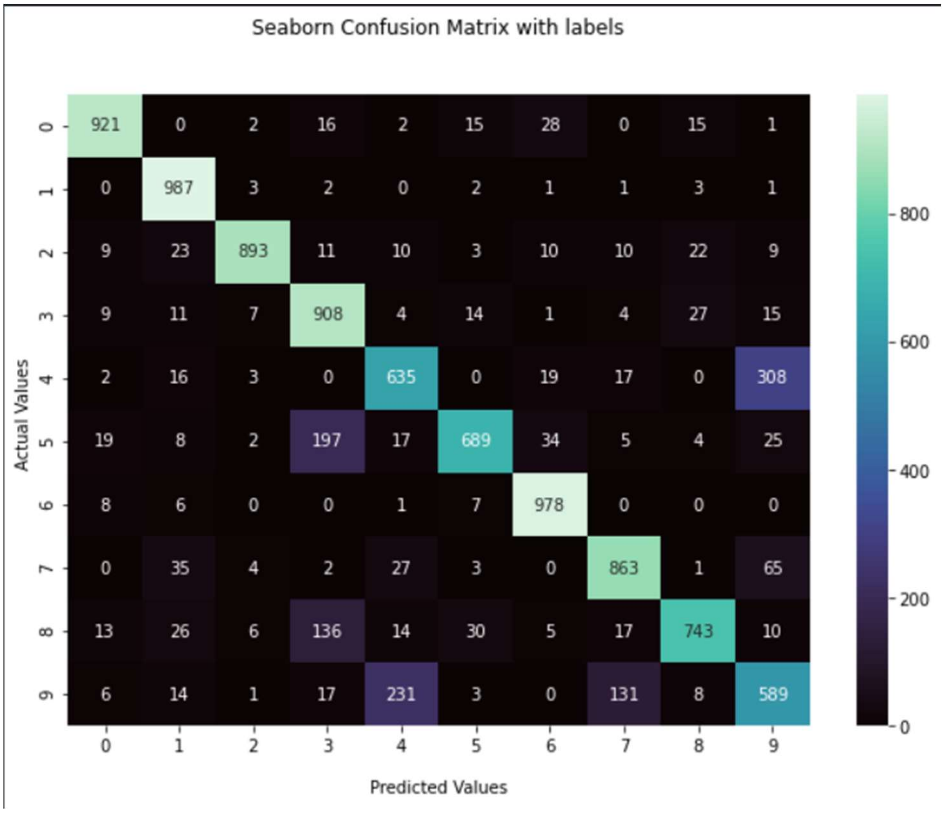
1. the decision region (how spread the region is), and
2. the penalty for misclassifying a data point

respectively.

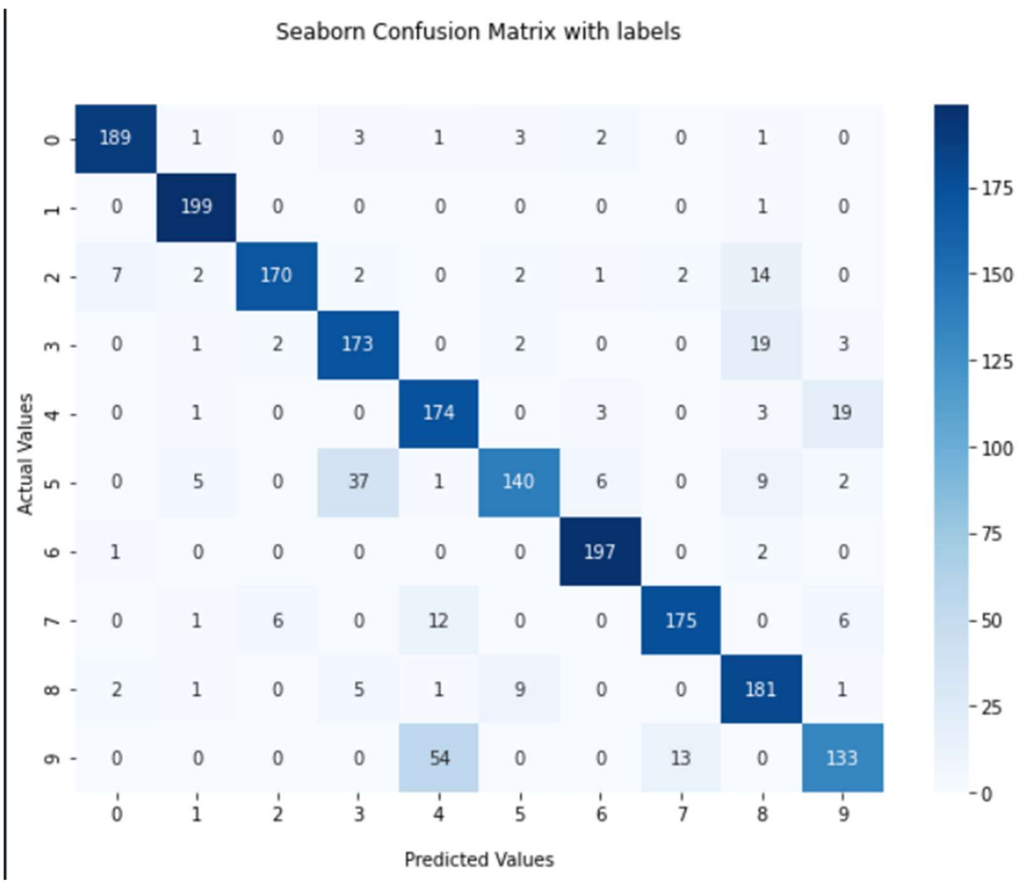
So, in contrast, it helps the SVM to become nonlinear rather than linear. RBF kernel function is similar to Normal distribution.

- For Kmeans-32 best model (PCA) confusion matrix:

Train:

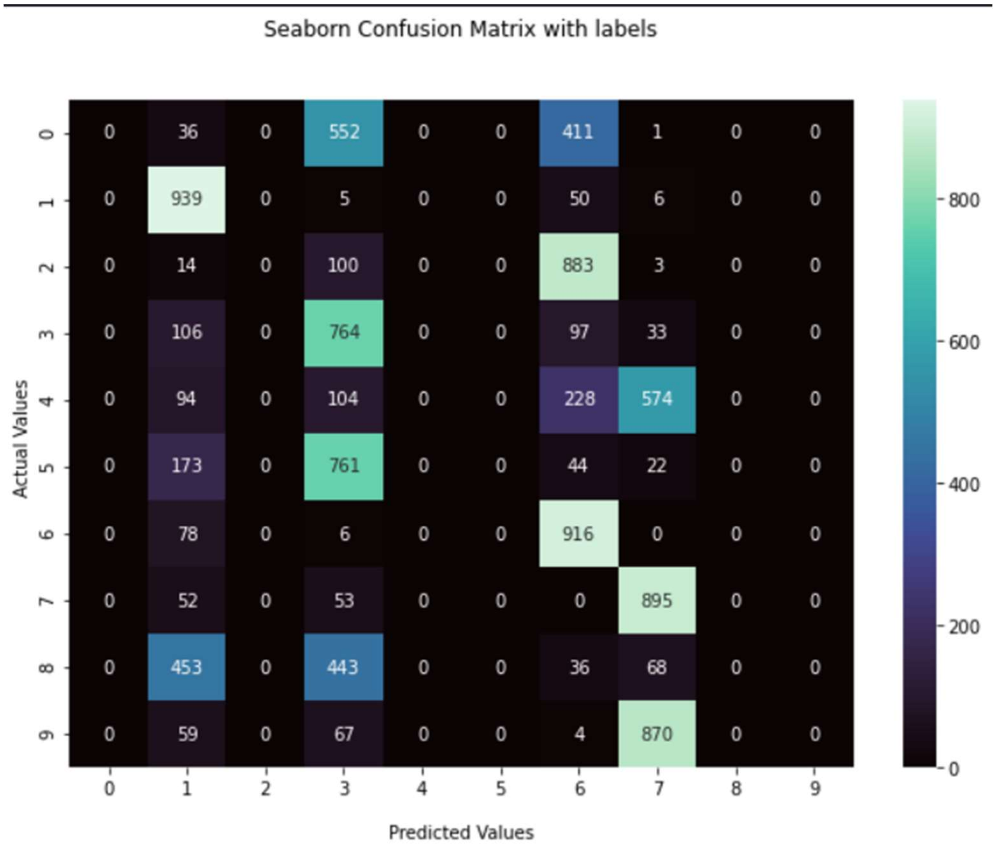


Test:

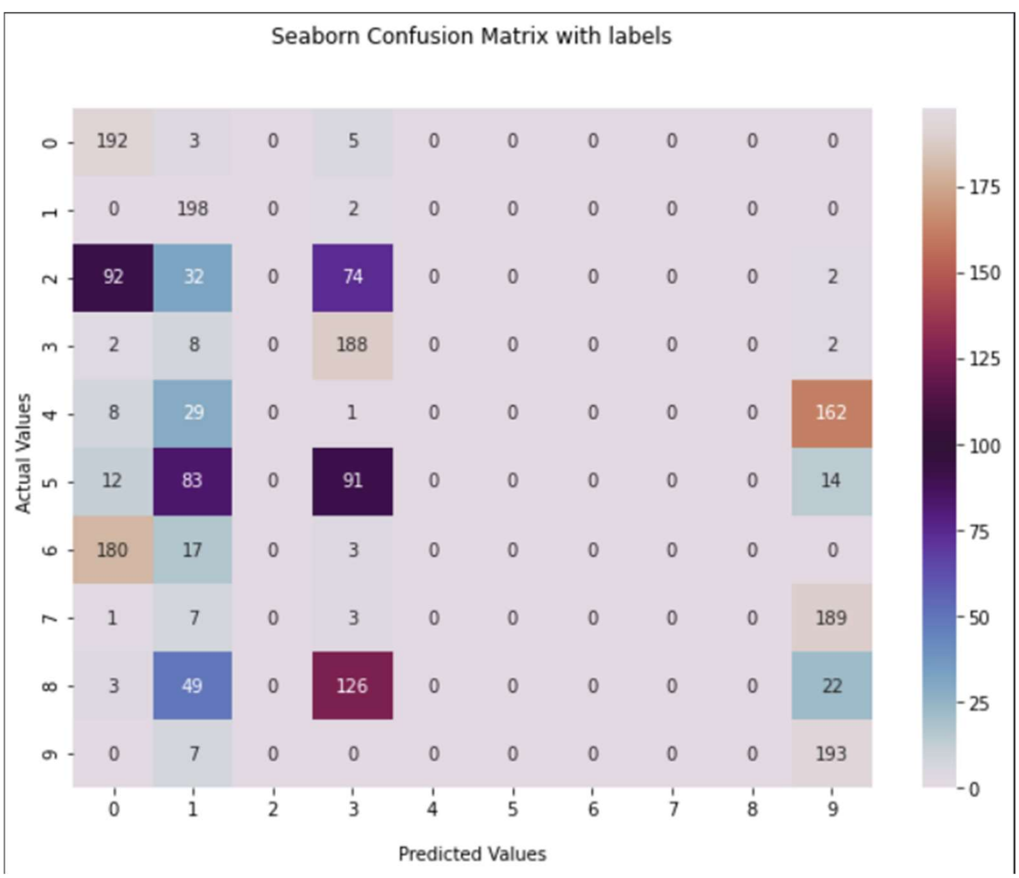


- For GMM-4 best model (PCA) confusion matrix:

Train:

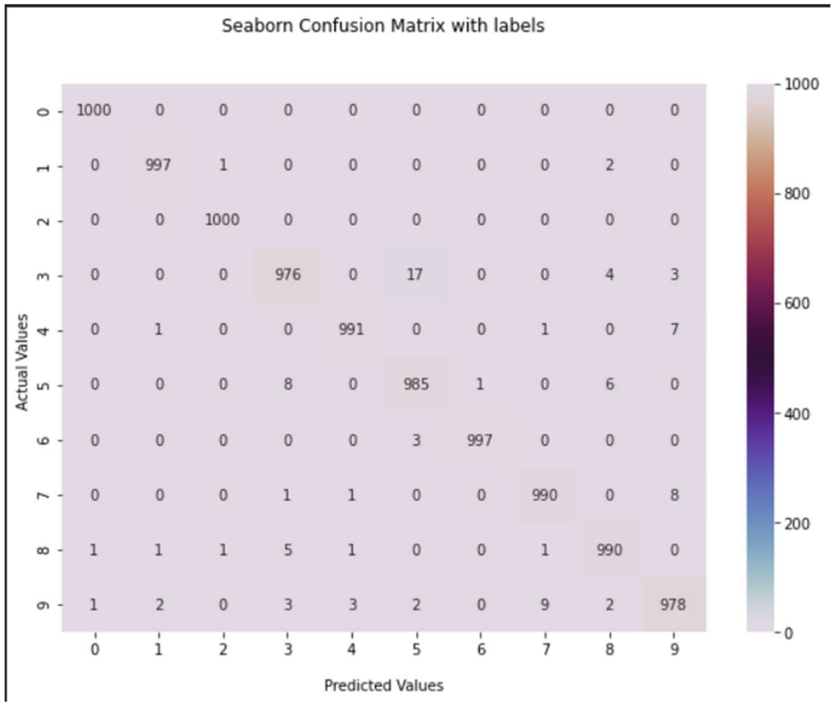


Test:

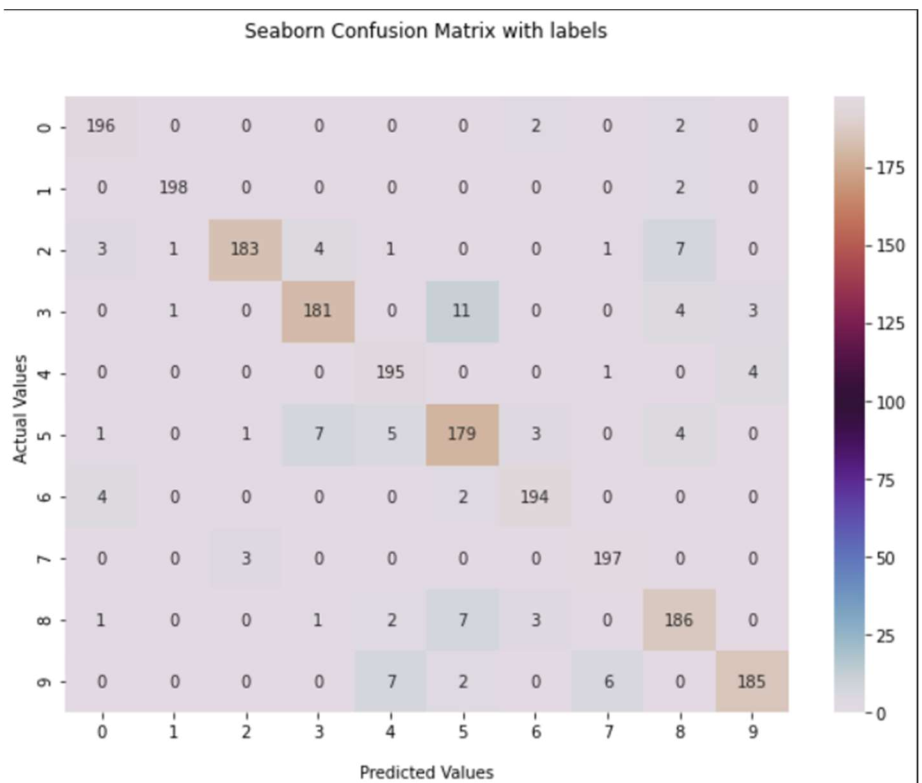


- For SVM, best training accuracy for Linear Kernel + Autoencoder

Train:

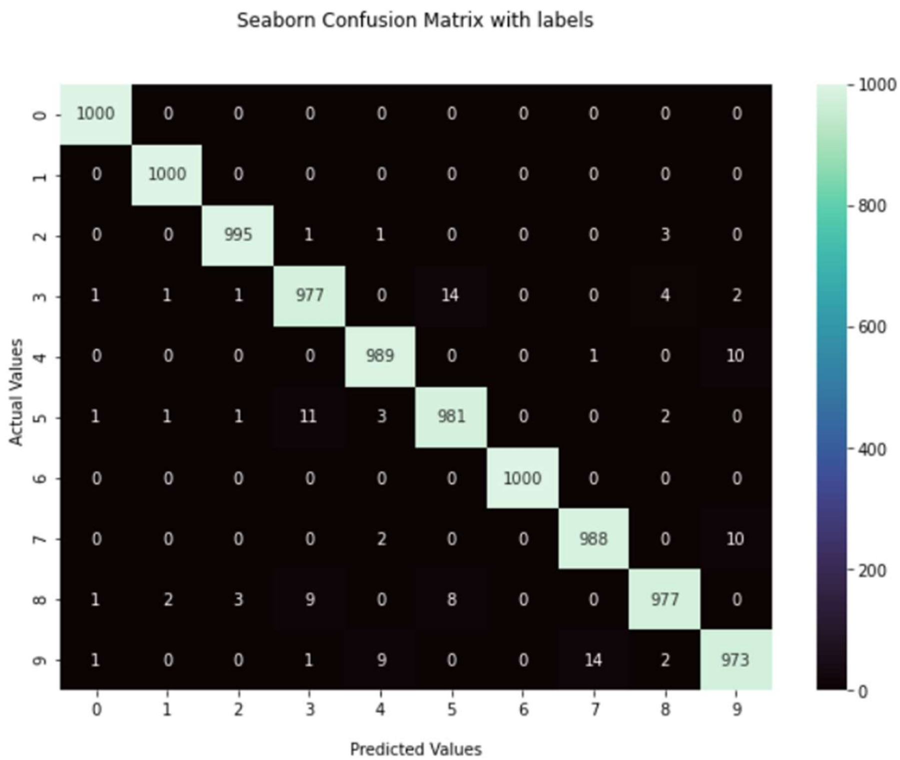


Test:

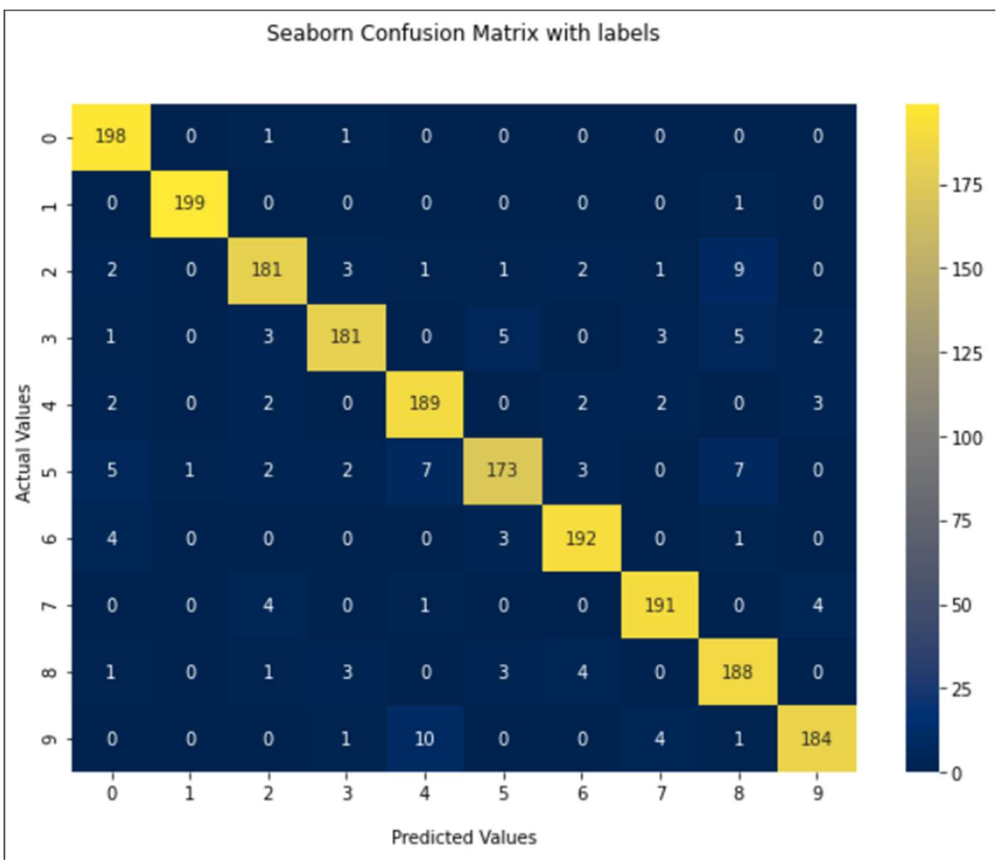


- But highest test accuracy for rbf Kernel + PCA

Train:



Test:



- Conclusion:

- As the number of clusters increases the accuracy increases, but of course it will be more computational expensive.
- As the number of Gaussian means increases the accuracy increases, but of course it will be more computational expensive, same as previous.
- It’s noted that KMeans is faster and performed better than Gaussian Mixture even though Gaussian Mixture is a more of generalization of Kmeans.
- Autoencoder takes much time because it’s not just a mathematical formula unlike other methods, besides choosing number of epochs, batch size, all of that affect the timing and processing speed. We can increase number of hidden layers or choose Convolutional layers but all of that besides it will require more time with more processing power (The training of the neural network used GPU not CPU), the better the network the worse the output, because the output may copy the input exactly and won’t learn many features, so we stick with simple models.
- For this high dimensional data, rbf filter was much better than linear filter and this was noted in the test accuracy but with almost around 2x *more or less* time required for computations, so does it worth this almost 2-3% increase? it depends.