**CO543: Image Processing**

**Lab 4**

Ranage R.D.P.R. - E/19/310

**Part 1**

**1. Using K-means algorithm Identify the different clusters of MNIST Handwritten Digits**

**a. Briefly describe the elbow method and the silhouette method**

**Elbow Method**: The elbow approach is a technique used to ascertain the most suitable number of clusters in a dataset. The concept is founded on the notion that as the quantity of clusters rises, the dispersion within each cluster diminishes. The elbow technique graphically represents the variation (or alternative metric for assessing clustering effectiveness) in relation to the number of clusters. The inflection point on the line, where the rate of decrease abruptly changes, reveals the appropriate number of clusters. This position signifies a compromise between the decrease in variability and the rise in the quantity of clusters. The appropriate number of clusters is typically determined by identifying the point at which the rate of reduction slows down significantly, resulting in a plot that resembles the shape of a "elbow".

**Silhouette Method:** The silhouette approach is a methodology employed to assess the caliber of clusters generated by a clustering algorithm. Cohesion refers to the degree of similarity between an object and its own cluster, whereas separation refers to the degree of dissimilarity between an object and other clusters. The silhouette score is a numerical measure that falls within the range of -1 to 1. A higher value suggests that the object is well suited to its own cluster and not well suited to nearby clusters. A score around 0 signifies the presence of clusters that overlap. The silhouette score is computed for each individual sample and subsequently averaged to produce a comprehensive score for the clustering process. A higher silhouette score indicates superior clustering. The silhouette approach aids in identifying the most suitable number of clusters by evaluating silhouette scores for various cluster numbers. The ideal decision is determined by selecting the number of clusters that yields the highest average silhouette score.

**b. Mention the criteria behind the way you define number of clusters**

The principle for determining the number of clusters is the elbow approach. The elbow method is a heuristic technique employed to ascertain the most suitable number of clusters in a given dataset. The process is creating a graph that shows the number of clusters (k) plotted against a measure of clustering quality, such as the within-cluster sum of squares or inertia. The objective is to identify the point on the graph where there is a significant change in the slope, known as the "elbow" point. The elbow point signifies the point at which there is a compromise between minimizing error (or variance) and maximizing the number of clusters. The term "diminishing returns" refers to the point at which the addition of more clusters no longer leads to a meaningful improvement in the quality of clustering. Thus, the ideal value for k is frequently determined by selecting the number of clusters at the elbow point.

**c. Visualize each cluster and justify the reasons for misclusted images(eg: 5 is in 8’s cluster).**

import cv2

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

#Loading datasets

df\_train = pd.read\_csv('train.csv')

df\_test = pd.read\_csv('test.csv')

df\_train.head()

**A screenshot of a computer

Description automatically generated**

df\_test.head()

**A screenshot of a computer

Description automatically generated**

#since training data set has a label column, lets drop it

y\_train = df\_train['label']

df\_train.drop(['label'],axis=1, inplace=True)

#defining the test and training dataframes

#training set

x\_train = df\_train.to\_numpy()

y\_train = y\_train.to\_numpy()

x\_test = df\_test.to\_numpy()

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

(42000, 784)

(42000,)

(28000, 784)

#lets take a look at few images along with the label

plt.figure(figsize=(12, 10))

for i in range(12):

    plt.subplot(3, 4, i + 1)

    plt.imshow(x\_train[i + 10].reshape(28, 28), cmap='gray')  # Reshape each image to 28x28

    plt.title(y\_train[i + 10])  # Show the corresponding label

    plt.axis('off')  # Turn off axis

plt.show()

**A number in black squares

Description automatically generated with medium confidence**

from sklearn.cluster import KMeans

k\_values = range(1, 25)  # Try different numbers of clusters from 1 to 10

inertia\_values = []

# Iterate over each value of k

for k in k\_values:

    kmeans = KMeans(n\_clusters=k, random\_state=23)

    kmeans.fit(x\_train)

    inertia\_values.append(kmeans.inertia\_)

plt.plot(k\_values, inertia\_values, marker='o')

plt.title('Elbow Method')

plt.xlabel('Number of clusters (k)')

plt.ylabel('Within-cluster sum of squares (Inertia)')

plt.xticks(k\_values)

plt.show()

**A graph of a number of clusters

Description automatically generated**

from sklearn.neighbors import KNeighborsClassifier

KMeans\_model = KNeighborsClassifier(n\_neighbors=10)

KMeans\_model.fit(x\_train,y\_train)

subset\_x\_test = x\_test[1230:1246]

predictions = KMeans\_model.predict(subset\_x\_test)

images = subset\_x\_test.reshape(16, 28, 28)

plt.figure(figsize=(12, 12))

for i in range(16):

    plt.subplot(4, 4, i + 1)

    plt.imshow(images[i], cmap='gray')

    plt.title("Predictions = " + str(predictions[i]))

    plt.axis('off')

plt.show()

**A collage of numbers

Description automatically generated**

KMeans\_model\_2 = KMeans(n\_clusters=10, random\_state=0).fit(x\_train)

cluster\_list = {i : np.where(KMeans\_model\_2.labels\_ == i) for i in range(15)}

for i in range(10):

  plt.figure(figsize=(10,10))

  for j in range(min(10, len(cluster\_list[i][0]))):

    plt.subplot(1,10,j+1)

    plt.imshow(x\_train[cluster\_list[i][0][j]].reshape(28,28), cmap='gray')

    plt.axis('off')



        

Misclassifications in K-means clustering of MNIST images occur because the algorithm groups images based on pixel similarity, causing different digits with similar pixel patterns to be assigned to the same cluster. The centroid, representing an average of all images in a cluster, may not distinctly represent any single digit, leading to overlaps. Additionally, the high dimensionality of the data (784 pixels) makes distance measures less effective, resulting in further overlaps. For example, a faintly written '3' is classified as an '8' here becuase their pixel patterns are similar.

**d. Suggest the ways to reduce the cluster errors.**

In order to mitigate cluster errors in K-means clustering, particularly when dealing with high-dimensional data such as MNIST, there are numerous tactics that can be employed. Utilizing dimensionality reduction methods such as PCA or autoencoders can effectively emphasize important patterns. Advanced clustering methods, such as Gaussian Mixture Models (GMM) or DBSCAN, provide increased flexibility and resilience. Thorough preprocessing, which involves normalizing and removing noise, guarantees that all features make equal contributions. Utilizing more effective initialization techniques such as K-means++ and evaluating the quality of clusters using measures like the Silhouette Score aids in identifying the most ideal clusters. Utilizing data augmentation methods for photos and employing post-processing techniques such as hierarchical clustering can enhance and improve the accuracy of the outcomes. The integration of these methodologies improves the precision of clustering and minimizes errors.

**Part 02**

**2. Using Artificial neural network and convolutional neural network Identify the different classes of MNIST Fashion dataset.**

**a. Initially train a classifier using artificial neural network while treating pixels as different features**

# improting libraries

import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tensorflow.keras import layers,models

from keras.models import Sequential

from keras.layers import Flatten, Dense

# Importing the dataset

df\_train = pd.read\_csv('fashion-mnist\_train.csv')

df\_test = pd.read\_csv('fashion-mnist\_test.csv')

df\_train.head()

**A screenshot of a cell phone

Description automatically generated**

df\_test.head()

**A screenshot of a computer

Description automatically generated**

y\_train = df\_train['label']

y\_test = df\_test['label']

df\_train.drop(['label'],axis=1, inplace=True)

df\_test.drop(['label'],axis=1, inplace=True)

x\_train = df\_train.to\_numpy()

y\_train = y\_train.to\_numpy()

x\_test = df\_test.to\_numpy()

y\_test = y\_test.to\_numpy()

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28).astype('float32')/255

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28).astype('float32')/255

class\_names = ['T\_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train\_preview = x\_train[85:100].reshape(15,28,28)

plt.figure(figsize=(10,7))

for i in range(15):

  plt.subplot(3,5,i+1)

  plt.xticks([])

  plt.yticks([])

  plt.imshow(train\_preview[i],  cmap=plt.cm.binary)

  plt.title('Label:{}'.format(class\_names[y\_train[i+85]]))

**A collage of different clothes

Description automatically generated**

model = Sequential()

model.add(Flatten(input\_shape=((28,28))))

model.add(Dense(100,activation = "relu"))

model.add(Dense(10,activation="softmax"))

model.compile(optimizer = "adam",loss="sparse\_categorical\_crossentropy",metrics=["accuracy"])

history = model.fit(x\_train, y\_train, epochs = 30, validation\_split=0.2, verbose=1)

A screenshot of a computer

Description automatically generated

def eval\_accuracy\_loss(history):

    # Get parameters for the testing set

    accuracy\_testing = history.history['accuracy']

    loss\_testing = history.history['loss']

    number\_of\_epochs = range(len(accuracy\_testing))

    # Get parameters for the validation set

    accuracy\_validation = history.history['val\_accuracy']

    loss\_validation = history.history['val\_loss']

    # Plotting the accuracy

    plt.figure(figsize=(10, 7))

    plt.plot(number\_of\_epochs, accuracy\_testing, 'b--', label='Training Accuracy')

    plt.plot(number\_of\_epochs, accuracy\_validation, 'r-', label='Validation Accuracy')

    plt.title('Accuracy Variation')

    plt.xlabel('Epochs')

    plt.ylabel('Accuracy')

    plt.legend()

    # Plotting the loss

    plt.figure(figsize=(10, 7))

    plt.plot(number\_of\_epochs, loss\_testing, 'b--', label='Training Loss')

    plt.plot(number\_of\_epochs, loss\_validation, 'r-', label='Validation Loss')

    plt.title('Loss Variation')

    plt.xlabel('Epochs')

    plt.ylabel('Loss')

    plt.legend()

eval\_accuracy\_loss(history)

A graph of a line graph

Description automatically generated

A graph with red and blue lines

Description automatically generated

#test the model with test data

pred = np.argmax(model.predict(x\_test[:20]), axis=1)

print(pred)

print(y\_test[:20])

A number on a white background

Description automatically generated

def test\_model(model):

    pred = model.predict(x\_test[120:135])

    pred = np.argmax(pred, axis=1)

    plt.figure(figsize=(15, 12))

    for i in range(15):

        plt.subplot(3, 5, i + 1)

        plt.imshow(x\_test[i+120], cmap="gray")

        plt.title(class\_names[y\_test[i+120]] + " | " + class\_names[pred[i]])

test\_model(model)

A collage of different types of clothing

Description automatically generated

**b. Train a Convolutional neural network(CNN) for the above data set considering data points as images.**

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=(3, 3), strides=(1, 1), padding='valid', activation='relu', input\_shape=(28, 28, 1)))

model.add(tf.keras.layers.MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))

model.add(tf.keras.layers.Dropout(rate=0.25))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(units=128, activation='relu'))

model.add(tf.keras.layers.Dense(units=10, activation='softmax'))

model.compile(loss=tf.keras.losses.sparse\_categorical\_crossentropy, optimizer=tf.keras.optimizers.Adam(), metrics=['accuracy'])

model.summary()

**A screenshot of a computer

Description automatically generated**

history\_cnn = model.fit(x\_train, y\_train,batch\_size=256,epochs=10, validation\_split=0.2,verbose=1)

**A table of numbers with black text

Description automatically generated with medium confidence**

eval\_accuracy\_loss(history\_cnn)

**A graph with a line

Description automatically generated**

**A graph with a line graph

Description automatically generated with medium confidence**

#test the model with test data

pred = np.argmax(model.predict(x\_test[:20]), axis=1)

print(pred)

print(y\_test[:20])

**A number on a white background

Description automatically generated**

test\_model(model)

**A collage of different types of clothing

Description automatically generated**

**c. Identify the difference between above 2 models**

Upon analysing the accuracy and loss curves of the ANN model, it becomes evident that the model starts to overfit the data during the training process after completing 3 epochs. This may be observed by analysing the validation and training curves after epoch=3, where it becomes evident that the disparity between them is progressively growing.

However, the CNN model does not follow this pattern. Since the accuracy and loss curves of both the training and validation datasets are consistently aligned and the discrepancy is not growing, it can be concluded that... Hence, the CNN model exhibits significantly higher accuracy in comparison to the ANN model.

**f. Discuss having more or less nodes in a single layer and having a deep or a shallow network against the computational complexity.**

Artificial Neural Networks (ANNs) are structured with an input layer, one or more hidden layers, and an output layer, each comprising neurons with associated weights and thresholds. Neurons activate when their output exceeds the threshold, transmitting data to subsequent layers. Adding layers increases neuron count and model complexity, potentially causing overfitting as some neurons fail to meet activation thresholds, leading to their removal. Dropout layers after hidden layers mitigate overfitting. ANNs may not suit large datasets due to explicit data point image detailing, demanding significant processing time. Contrastingly, Convolutional Neural Networks (CNNs) automatically extract image features, making them ideal for larger datasets. ANNs, favoring 1-dimensional vectors, exponentially increase parameters during training, raising memory and time complexities. Given CNNs' minimal human intervention and efficiency, they excel across various domains.

**g. Discuss about the way you defined the optimum neural network architecture for the above problem.**

An optimal strategy is to employ an equal number of nodes in each concealed layer.

Nevertheless, it is crucial to take into account the quantity of layers and nodes, as an excessive number of nodes can result in overfitting (when the model becomes excessively tailored to the training data), while a reduced number of nodes can lead to underfitting (when the model is too simplistic to capture patterns). Networks with a greater number of hidden layers (more than 2-3) can result in inaccurate models because to overfitting and a significant increase in computational complexity.