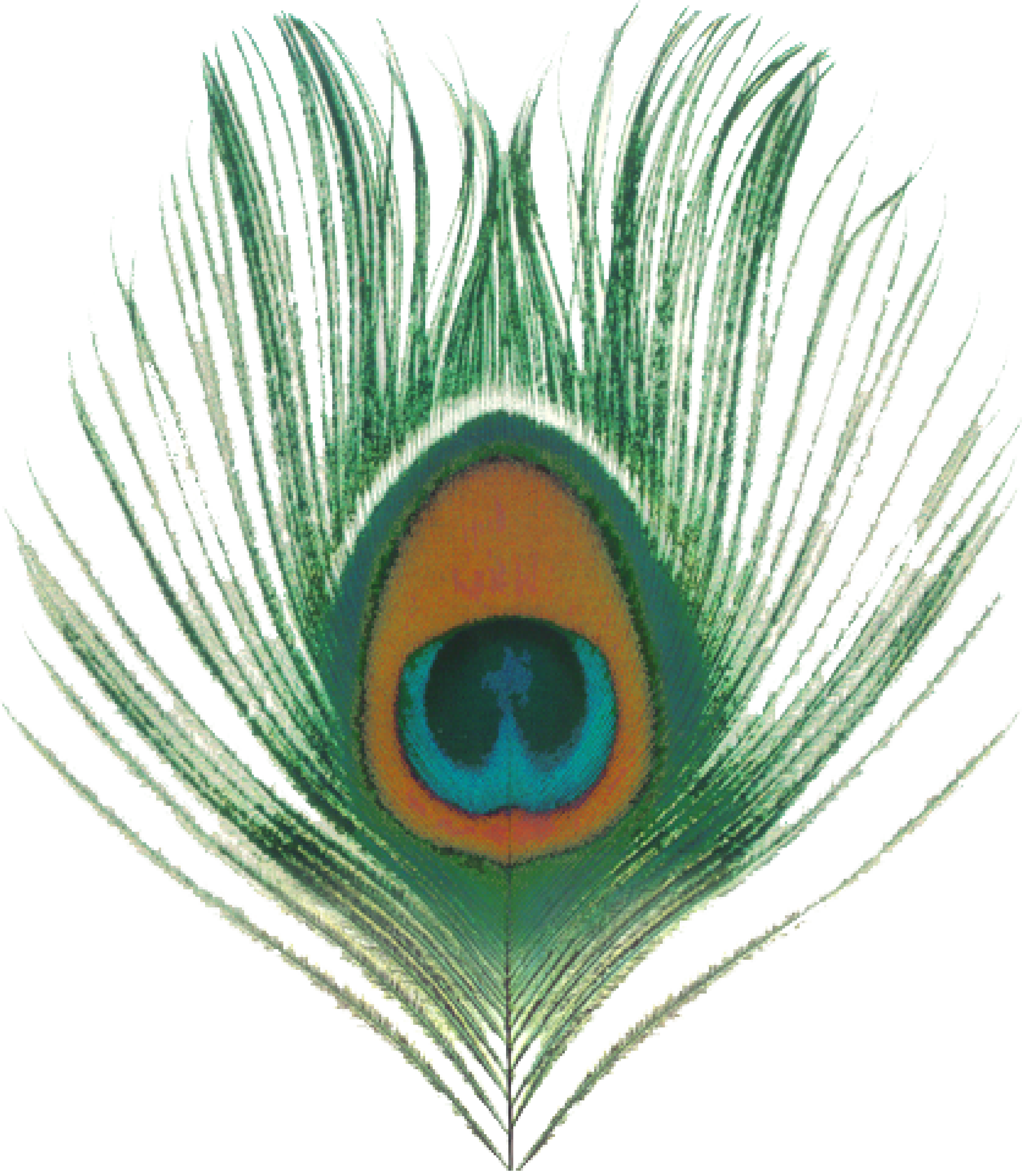
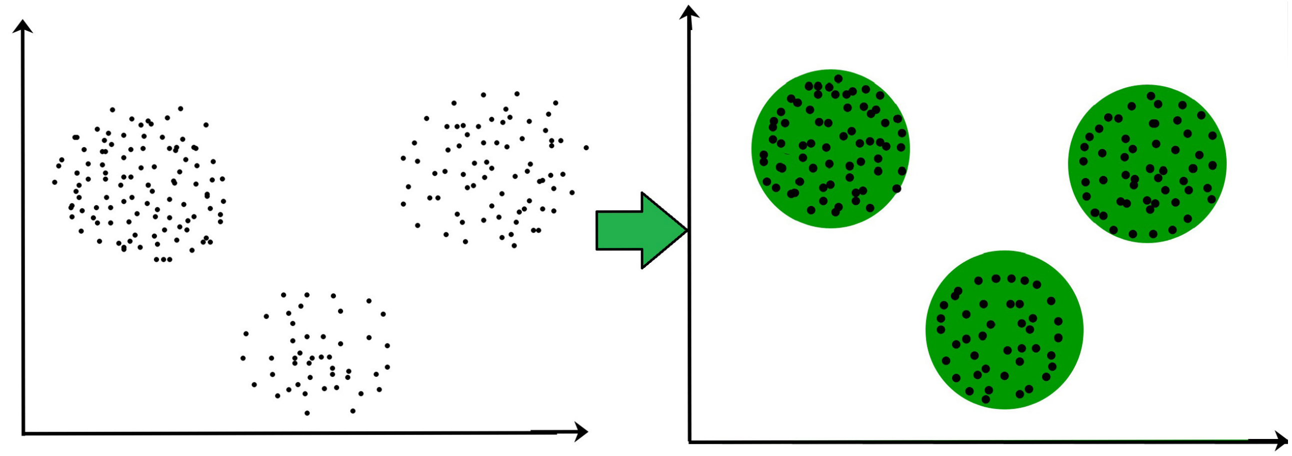
HIERARCHICAL CLUSTERING

ARIHARASUDHAN



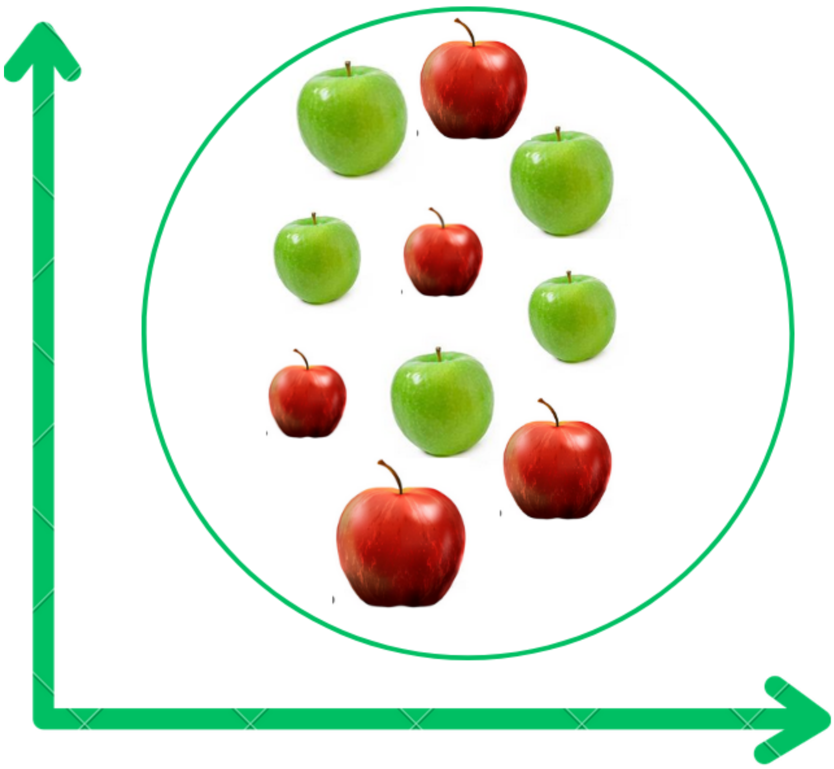
THE CLUSTERING

Ah... The World knows what it is.. Clustering is a well-known unsupervised algorithm that groups data points, which can be images or any type of data, into clusters. We have also encountered the Gaussian Mixture Model (GMM), which is a soft-clustering algorithm. GMM assigns probabilities to data points, indicating to which component or cluster they belong. GMM makes an assumption that the data distribution is a mixture of Gaussian distributions. Besides, K-Means clustering is widely known! All these algorithms somehow group the data into clusters as shown below.

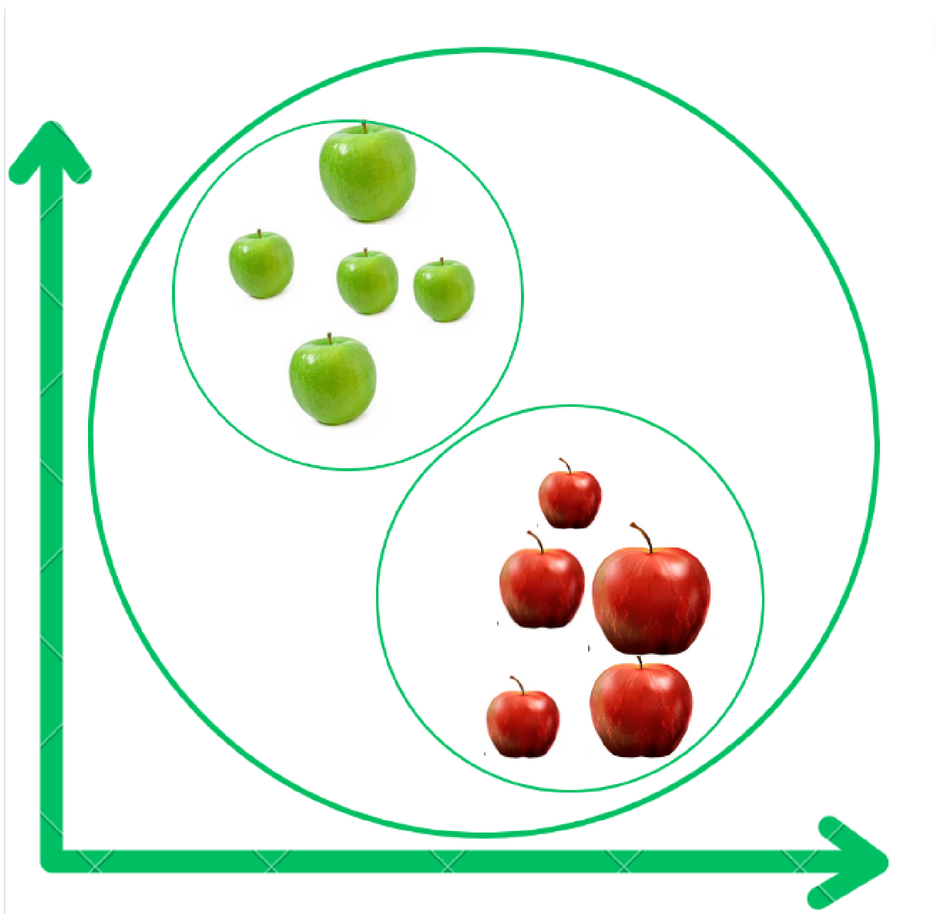


THE HIERARCHICAL CLUSTERING

Now, what if we are tasked with further clustering the already formed clusters? That's where hierarchical clustering comes in! For instance, if we've already clustered images into two groups representing bears and apples, The Apple Cluster might contain subclusters of green apples and red apples, similar to The Bear Cluster, which can contain subclusters of grizzly and panda bears. We now need to subcluster the initial clusters, creating subclusters for the Apple Cluster and The Bear Cluster. The Apple Cluster should be further divided into two subclusters, and the same goes for The Bear Cluster. How can we achieve this? Will it be efficient? Let’s discuss in the further pa(ss)ages. These following two padams show what we refer to.



PADAM 1 : THE APPLE CLUSTER



PADAM 2 : TWO SUBCLUSTERS IN APPLE CLUSTER

CLUSTERING METHODS

Clustering clusters into subclusters is a hierarchical clustering problem, which is a more advanced technique than traditional clustering. In our case, we have clusters like "apples," and we want to further divide them into subclusters like "green apples" and "red apples." To achieve this, we can use techniques like agglomerative hierarchical clustering or divisive hierarchical clustering. These are some commonly used techniques to perform hierarchical clustering. Ensure that we have data representing our initial clusters. Then, identify the features that will be used for subclustering. In our case, features could be color, size and other characteristics of the apples.

As we have disussed, two common methods are agglomerative and divisive clustering.

AGGLOMERATIVE CLUSTERING

☆ Start with each data point (individual apple) as a separate cluster.

☆ Iteratively merge clusters that are most similar based on our chosen similarity metric (e.g., Euclidean distance, cosine similarity).

☆ Continue merging clusters until we have the desired subclusters.

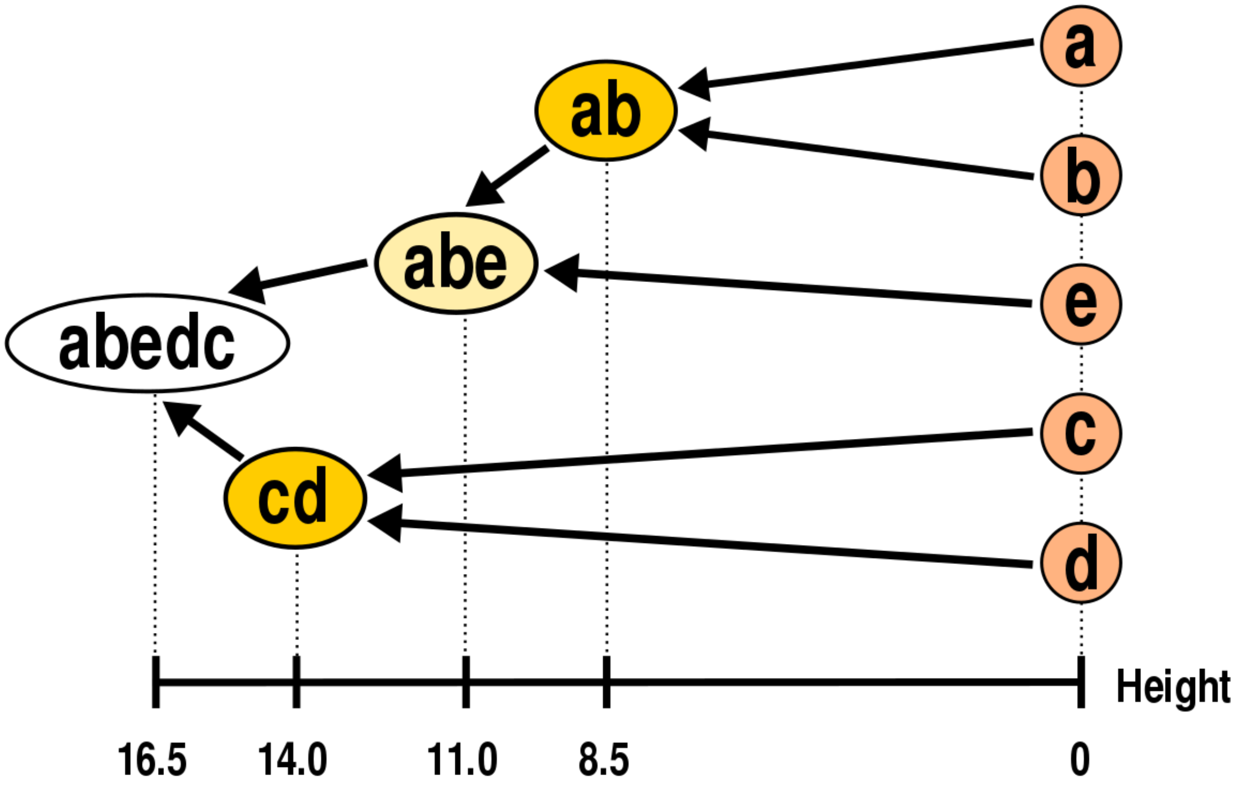
DIVISIVE CLUSTERING

☆ Start with all data points as one cluster.

☆ Iteratively split clusters into smaller subclusters based on the dissimilarity between data points.

☆ Continue splitting clusters until we have the desired subclusters.

We may need to specify the number of subclusters we want or use a stopping criterion for hierarchical clustering. Common approaches include cutting the hierarchical tree (dendrogram) at a certain height or using a specific number of clusters.

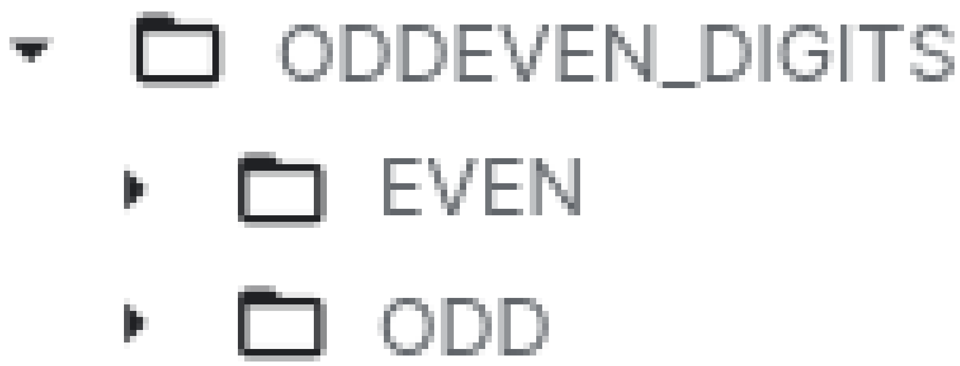


Use a dendrogram to visualize the hierarchical structure and subclusters. This will help you identify where to cut the tree to obtain our subclusters.

Finally, based on the cut we make in the dendrogram, assign each data point (apple) to a subcluster. Evaluate the quality of your subclusters using appropriate metrics and adjust the clustering parameters if necessary.

LET’S DO IT

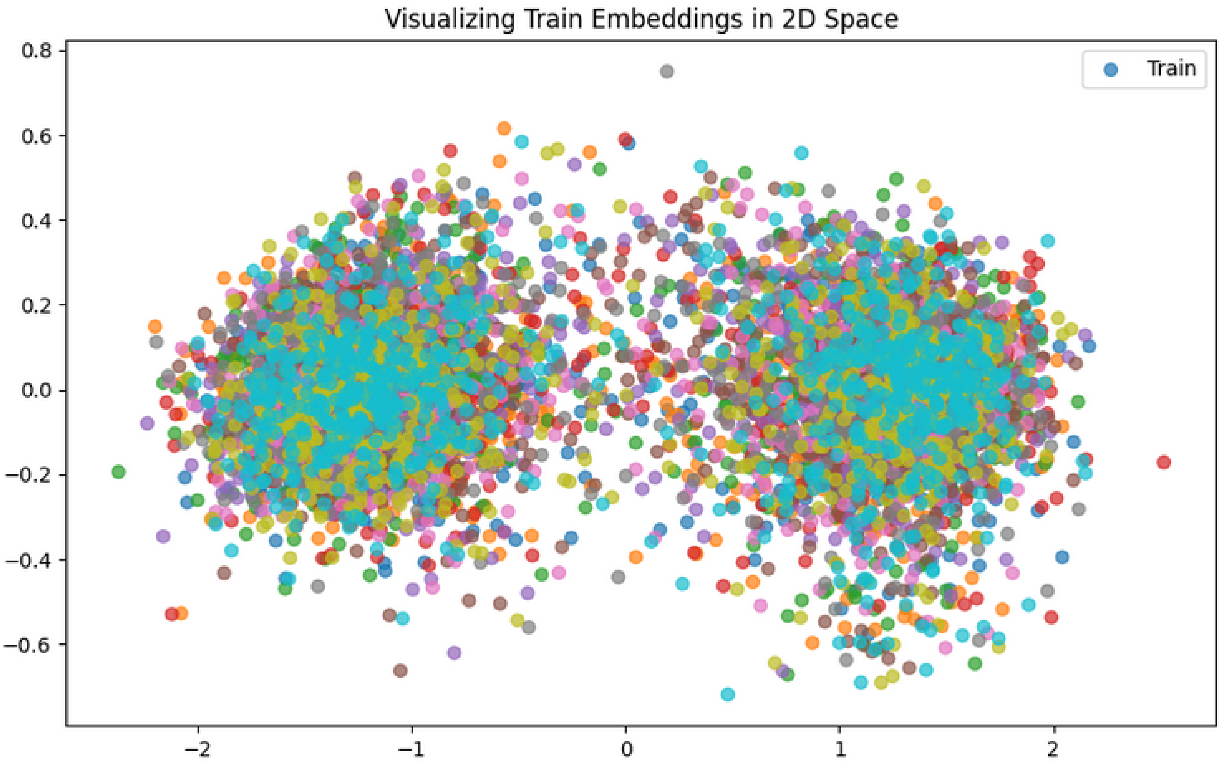
In order to understand it deeply, let me introduce one use-case. We have a datafolder consisting of the subfolders of ODD and EVEN. The ODD folder have images of MNIST ODD Digits like 1,3,5,7 and The EVEN folder have images of MNIST EVEN Digits like 2,4,6,8 as shown below.



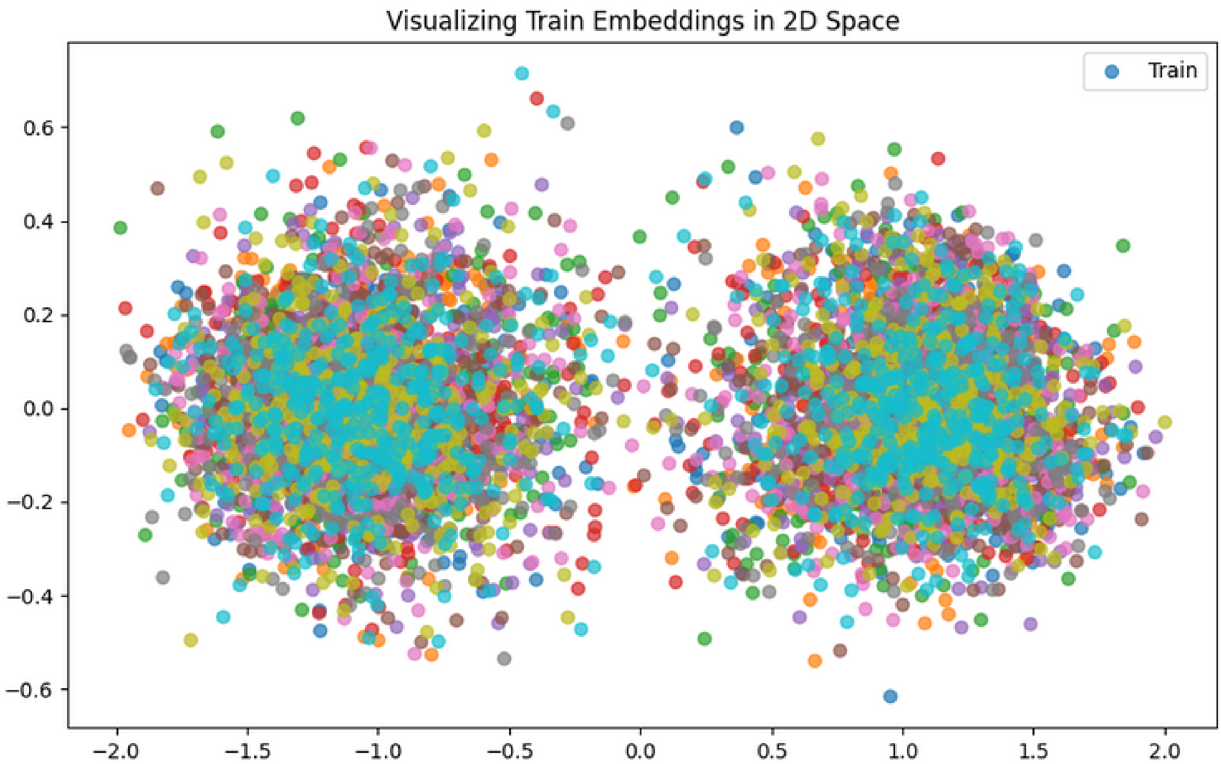
Now, we need to form two clusters. One should be the ODD Cluster and another one should be the EVEN Cluster. Let’s use a triplet loss training to efficiently make two clusters. I use the following simple CNN Model to extract image embeddings.

```
class EmbeddingNet(nn.Module):
    def __init__(self):
        super(EmbeddingNet, self).__init__()
        self.convnet = nn.Sequential(
     nn.Conv2d(1, 32, 5), nn.PreLU(), nn.MaxPool2d(2, stride=2),
     nn.Conv2d(32, 64, 5), nn.PReLU(), nn.MaxPool2d(2, stride=2))
        self.fc = nn.Sequential(
            nn.Linear(64 * 53 * 53, 256),
            nn.PReLU(),
            nn.Linear(256, 256),
            nn.PReLU(),
            nn.Linear(256, 2))
    def forward(self, x):
        output = self.convnet(x)
        output = output.view(output.size()[0], -1)
        output = self.fc(output)
        return output
    def get_embedding(self, x):
        return self.forward(x)
```

On 10^{th} epoch, the clusters were formed as shown below.



On 15^{th} epoch, the clusters were formed like,



So, now we have two clusters representing The ODD and EVEN classes. We want to perform subclustering... The ODD Cluster itself has the embeddings of images in the subfolders 1,3,5 & 7. Similarly, The EVEN Cluster has the embeddings of images in the subfolders 2,4,6 & 8. Now, we have to perform a hierarchical clustering that clusters the formed clusters into n classes. In our case, both ODD and EVEN clusters have images of 4 classes such as 1,3,5 & 7 and 2,4,6 & 8 respectively. So, we need to subcluster the formed two clusters into 8 sub-clusters.

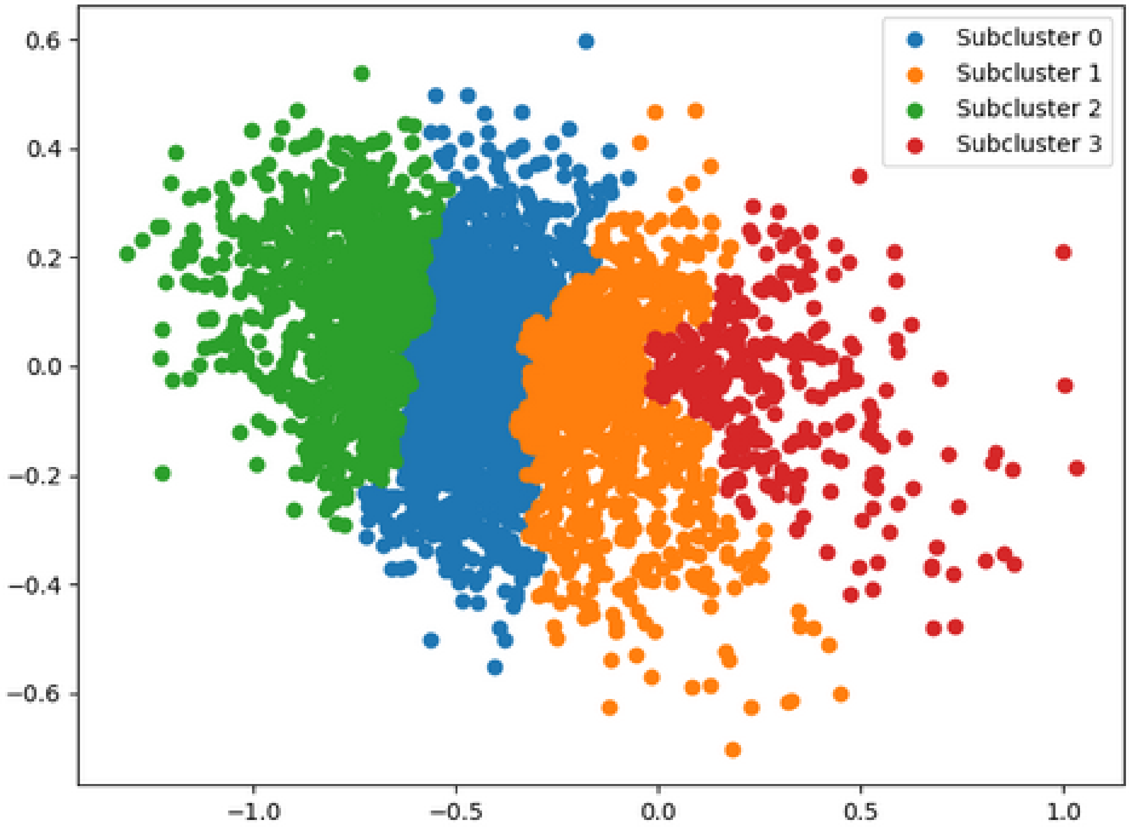
```
import numpy as np
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt

train_embs = train_embs.cpu()
odd_cluster_indices = [i for i, label in enumerate(train_labels) if label[0] == "ODD"]
even_cluster_indices = [i for i, label in enumerate(train_labels) if label[0] == "EVEN"]

# Create a function to visualize subclusters
def visualize_subclusters(data, sublabels, title):
    unique_labels = np.unique(sublabels)
    plt.figure(figsize=(8, 6))
    for label in unique_labels:
        plt.scatter(data[sublabels == label][:, 0], data[sublabels == label][:, 1], label=f'Subcluster {label}')
    plt.title(title)
    plt.legend()
    plt.show()
```

In the above-provided code, two lists, odd_cluster_indices and even_cluster_indices, are generated based on the train_labels array. Data points are divided into these two clusters by checking the initial characters of their labels. Data points with labels starting with ODD are placed in the odd_cluster_indices list, while those starting with "EVEN" are placed in the even_cluster_indices list. This separation serves as the basis for subsequent clustering and analysis of data points belonging to "ODD" and "EVEN" clusters.
Next step is the important one.

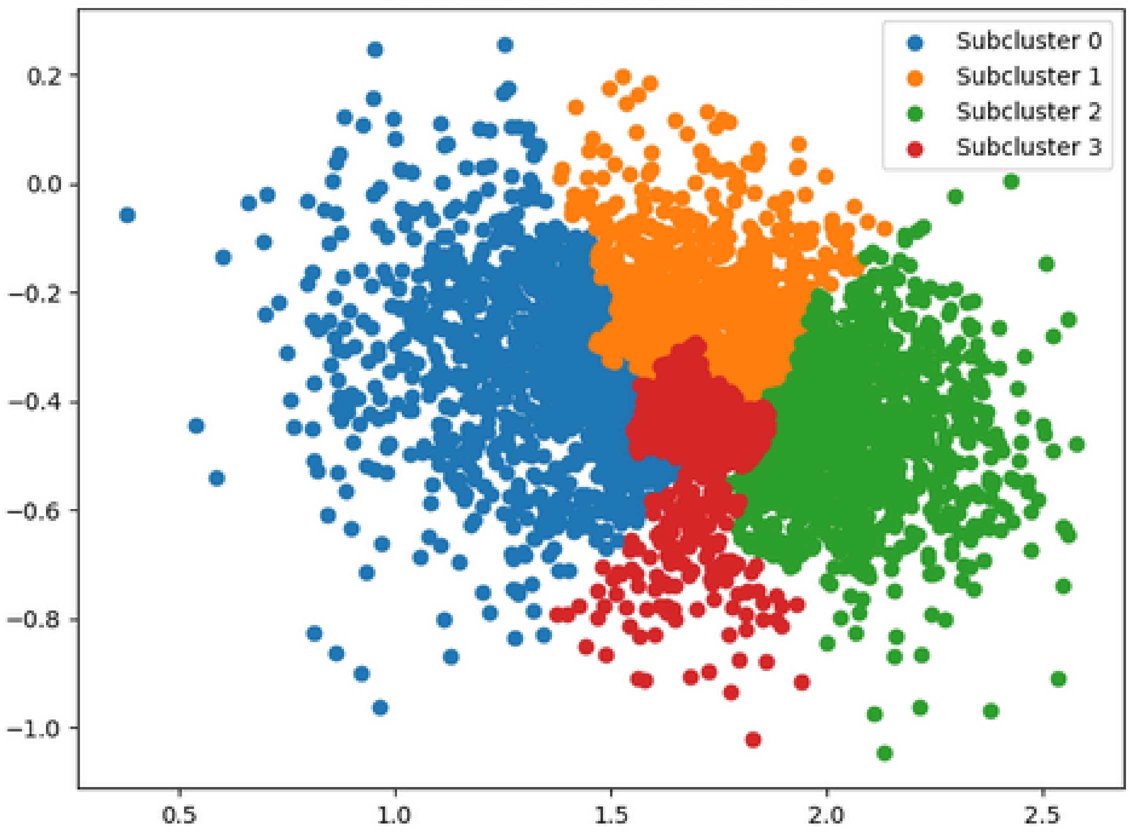
```
# Check if there are data points in the clusters
if len(odd_cluster_indices) > 0:
    odd_cluster_data = train_embs[odd_cluster_indices].cpu()
    n_subclusters_odd = 4  # Number of subclusters for Odd cluster
    agglomerative_odd = AgglomerativeClustering(n_clusters=n_subclusters_odd, metric='euclidean', linkage='ward')
    odd_cluster_sublabels = agglomerative_odd.fit_predict(odd_cluster_data)
    visualize_subclusters(odd_cluster_data, odd_cluster_sublabels, "Subclusters within Odd Cluster")

if len(even_cluster_indices) > 0:
    even_cluster_data = train_embs[even_cluster_indices].cpu()
    n_subclusters_even = 4  # Number of subclusters for Even cluster
    agglomerative_even = AgglomerativeClustering(n_clusters=n_subclusters_even, metric='euclidean', linkage='ward')
    even_cluster_sublabels = agglomerative_even.fit_predict(even_cluster_data)
    visualize_subclusters(even_cluster_data, even_cluster_sublabels, "Subclusters within Even Cluster")
```

In this section of the code, the script checks if there are any data points in the "ODD" and "EVEN" clusters by verifying the lengths of the odd_cluster_indices and even_cluster_indices lists. If data points exist in either cluster, further analysis is conducted. For the "ODD" cluster, an Agglomerative Clustering algorithm is applied with the specified number of subclusters (n_subclusters_odd=4) and using the Euclidean distance metric with the Ward linkage method. This creates subcluster assignments within the ODD cluster, which are then visualized using the visualize_subclusters function. The same process is repeated for the EVEN cluster. This code allows for hierarchical clustering within the "ODD" and "EVEN" clusters.

Subclusters within Odd Cluster



Subclusters within Even Cluster



We can incorporate A NeuralNet to refine the formed sub-clusters.

MERCI