rock-data-classificaion

October 27, 2024

[3]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[4]:
```python
import os
import numpy as np
from PIL import Image

# Define your image folder path
folder_path = '/content/drive/MyDrive/360 Rocks'


images = []
filenames = []

standard_size = (64, 64)

# Loop through each file in the folder
for filename in os.listdir(folder_path):
    if filename.lower().endswith(('.jpg')):

        file_path = os.path.join(folder_path, filename)

        # Load the image, convert to grayscale, and resize
        img = Image.open(file_path).convert('L')
        img_resized = img.resize(standard_size)


        img_data = np.array(img_resized).flatten()


        images.append(img_data)
        filenames.append(filename)


images_array = np.array(images)
```

```
print(images_array.shape)
```

```
(360, 4096)
```

1. Applied PCA to the images from folder '360 Rocks'to preserve 90% of the variance

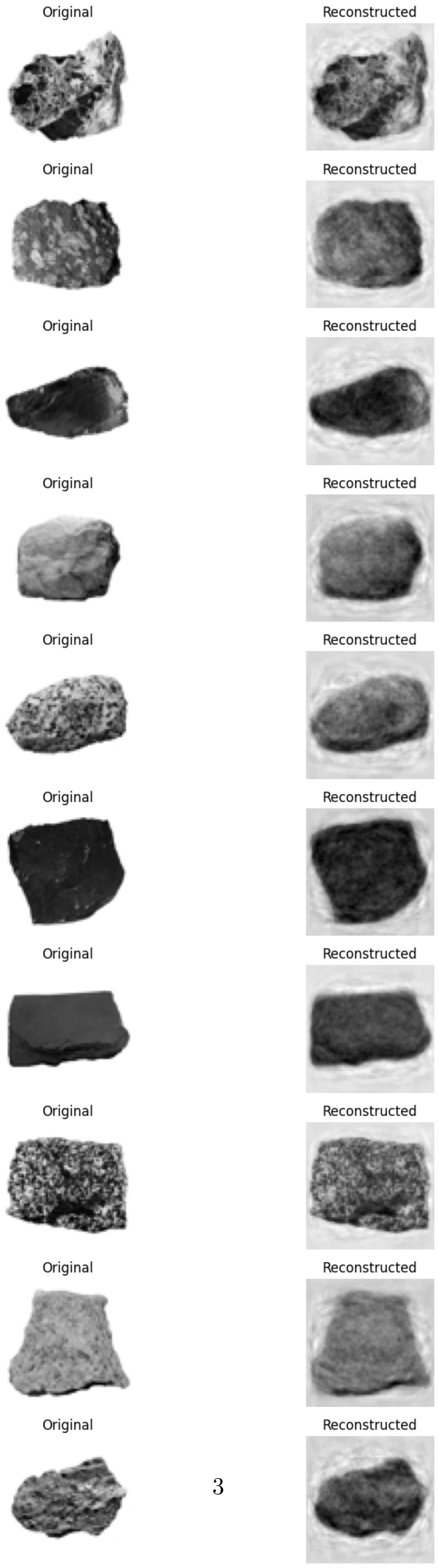
[5]:
```
from sklearn.decomposition import PCA

# Apply PCA with n_components set to 0.9 to preserve 90% of the variance
pca = PCA(n_components=0.9)
images_pca = pca.fit_transform(images_array)

# The number of components needed to preserve 90% of variance
num_components_90_var = pca.n_components_
print(num_components_90_var)
```

```
93
```

Accoring to the n.components_ variable from the PCA class, the number of components needed to preserve 90% of the variance is 93.

2. Plotted 10 images in the original form (without PCA) and then plotted their reconstruction (projection in the original space) after having 90% of variance using PCA.

[6]:
```
import matplotlib.pyplot as plt

np.random.seed(42)
indices = np.random.choice(images_array.shape[0], 10, replace=False)

images_reconstructed = pca.inverse_transform(images_pca[indices])

# Plot original and reconstructed images side by side
fig, axes = plt.subplots(10, 2, figsize=(10, 20))
for i, index in enumerate(indices):
    # Original image
    axes[i, 0].imshow(images_array[index].reshape(64, 64), cmap='gray')
    axes[i, 0].set_title('Original')
    axes[i, 0].axis('off')

    # Reconstructed image
    axes[i, 1].imshow(images_reconstructed[i].reshape(64, 64), cmap='gray')
    axes[i, 1].set_title('Reconstructed')
    axes[i, 1].axis('off')
plt.tight_layout()
plt.show()
```


Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed

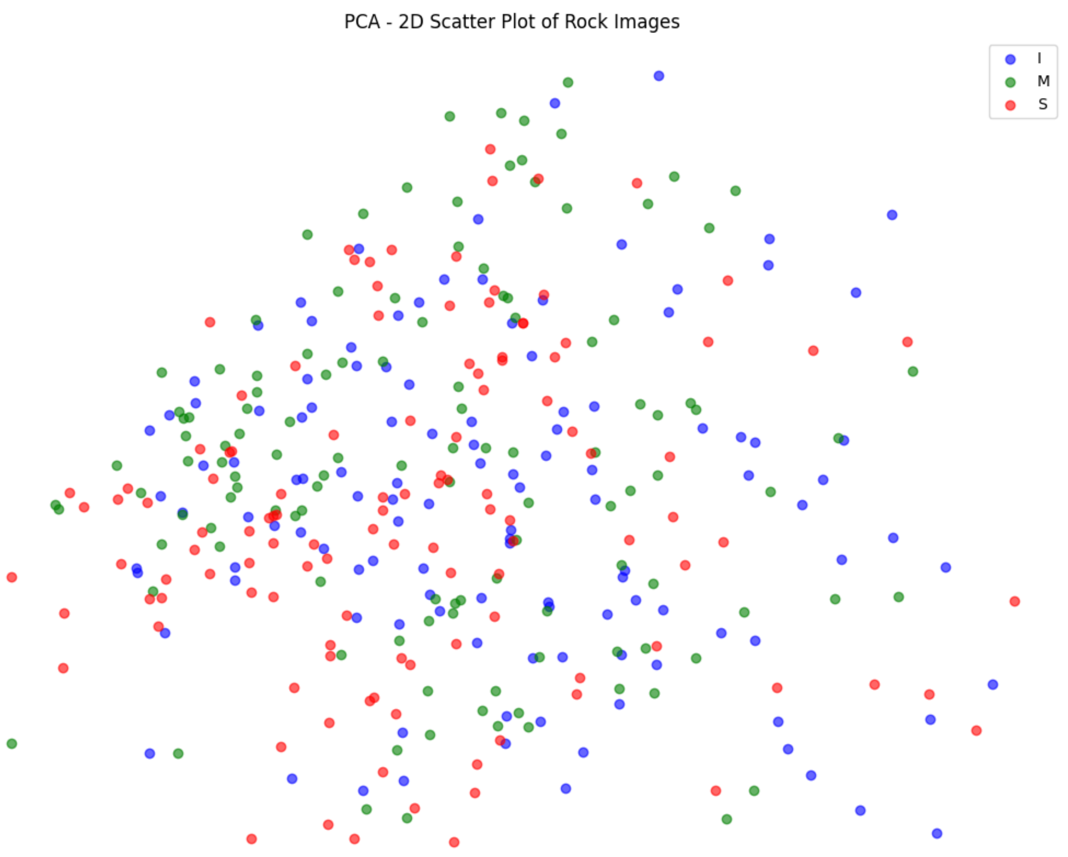
3. Used PCA to reduce dimensionality to only 2 dimensions, to check how much of the variance is explained with the first two principal components?

[7]:
```
from sklearn.decomposition import PCA

pca_2d = PCA(n_components=2)
images_pca_2d = pca_2d.fit_transform(images_array)

# Calculate the variance explained by the first two components
variance_explained_2d = np.sum(pca_2d.explained_variance_ratio_)

print(f"Variance explained by the first two principal components:␣
↪{variance_explained_2d*100:.2f}%")
```

```
Variance explained by the first two principal components: 38.13%
```

Plot a 2D scatter plot of the images spanned by the first two principal components. Each image will be represented with a dot. Make the color of the dot correspond to the image category (so you will have three different colors). Then add some rock images to the visualization to better understand what features in the images are accounting for the majority of variance in the data. Repeat the process and create the same type of plots for t-SNE, LLE and MDS.

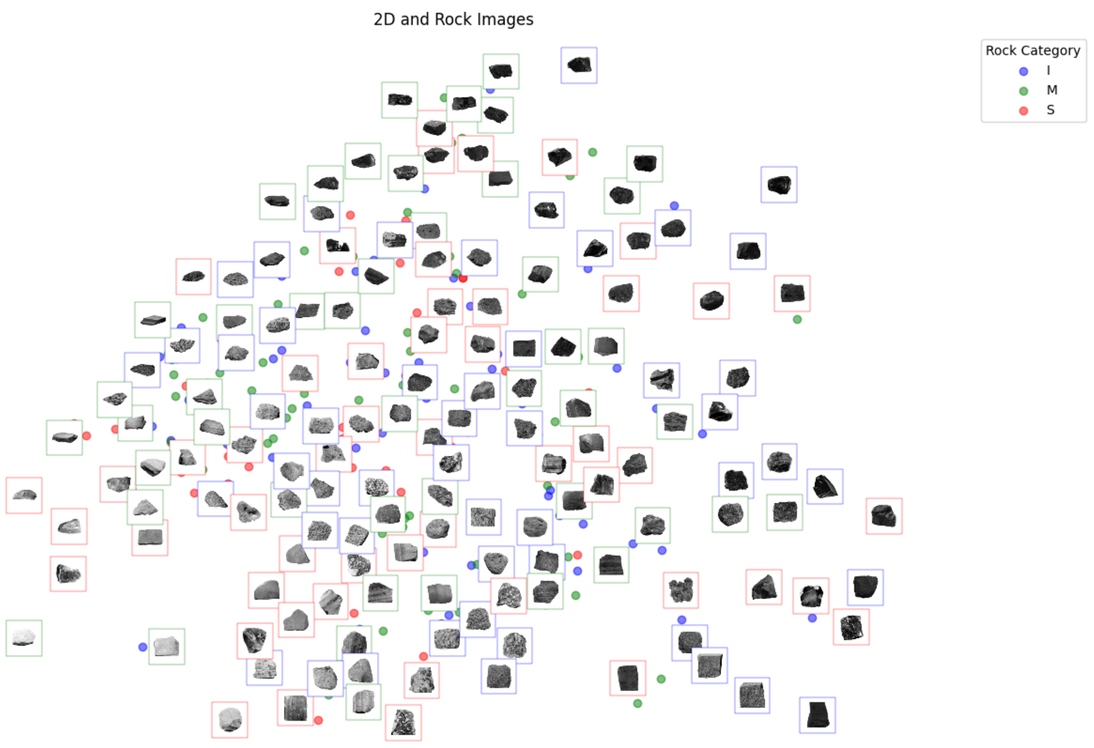
[8]:
```
categories = np.array([filename[0] for filename in filenames])  # Assumes␣
↪filenames list is available

# Mapping categories to colors
category_colors = {'I': 'blue', 'M': 'green', 'S': 'red'}
colors = np.array([category_colors[category] for category in categories])

# Scatter plot
plt.figure(figsize=(13, 10))
for category, color in category_colors.items():
    ix = np.where(categories == category)
    plt.scatter(images_pca_2d[ix, 0], images_pca_2d[ix, 1], c=color,␣
↪label=category, alpha=0.6)
plt.title('PCA - 2D Scatter Plot of Rock Images')
plt.axis("off")
plt.legend()
plt.show()
```



Add rock images to 2D PCA Scatter plot

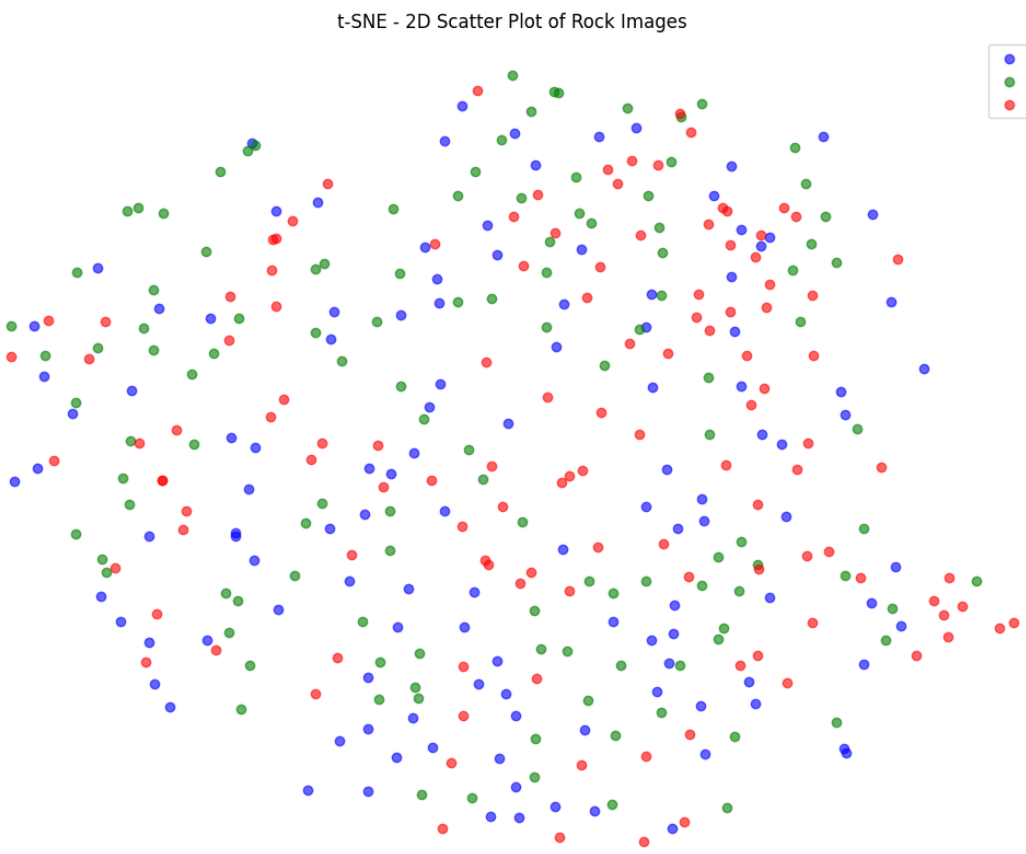
```
[9]: from sklearn.preprocessing import MinMaxScaler
     from matplotlib.offsetbox import AnnotationBbox, OffsetImage

     def plot_rocks(X, categories, min_distance=0.04, images=None, figsize=(13, 10),␣
      ↪image_zoom=0.1):
         # Normalize the PCA features to range from 0 to 1
         X_normalized = MinMaxScaler().fit_transform(X)

         # Define colors for each category
         category_colors = {
             'I': 'blue',
             'M': 'green',
             'S': 'red'
         }


         fig, ax = plt.subplots(figsize=figsize)
```

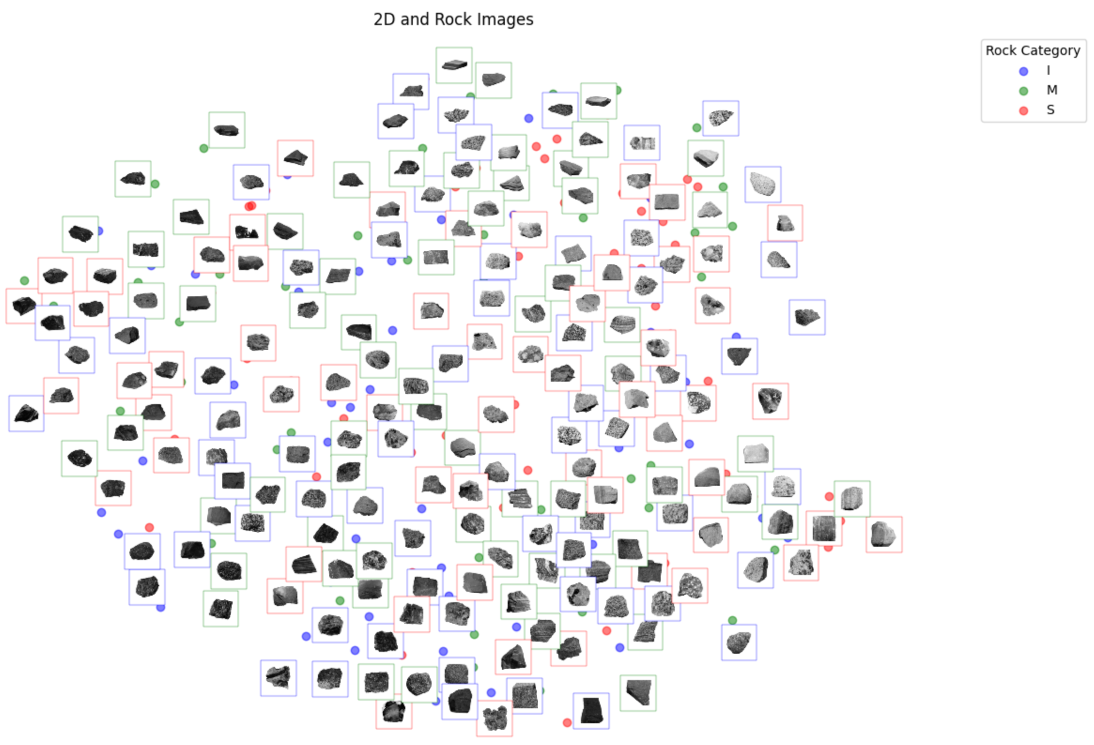
```
    for category in np.unique(categories):
        ax.scatter(X_normalized[categories == category, 0],
                   X_normalized[categories == category, 1],
                   c=category_colors[category], label=category, alpha=0.5)

    plt.axis("off")

    # Adding images with colored frame based on the category
    neighbors = np.array([[10., 10.]])
    for index, image_coord in enumerate(X_normalized):
        category = categories[index]
        closest_distance = np.linalg.norm(neighbors - image_coord, axis=1).min()
        if closest_distance > min_distance:
            neighbors = np.r_[neighbors, [image_coord]]
            if images is not None:
                image = images[index].reshape(64, 64)
                imagebox = AnnotationBbox(OffsetImage(image, zoom=image_zoom,␣
 ↪cmap="gray"),
                                          image_coord, frameon=True,
                                         ␣
 ↪bboxprops=dict(edgecolor=category_colors[category], linewidth=0.3))
                ax.add_artist(imagebox)


    plt.legend(title='Rock Category', bbox_to_anchor=(1.05, 1), loc='upper␣
 ↪left')
    plt.title('2D and Rock Images')
    plt.show()

plot_rocks(X=images_pca_2d, categories=categories, images=images_array,␣
 ↪image_zoom=0.3)
```



Repeat the same process for t-SNE, LLE and MDS

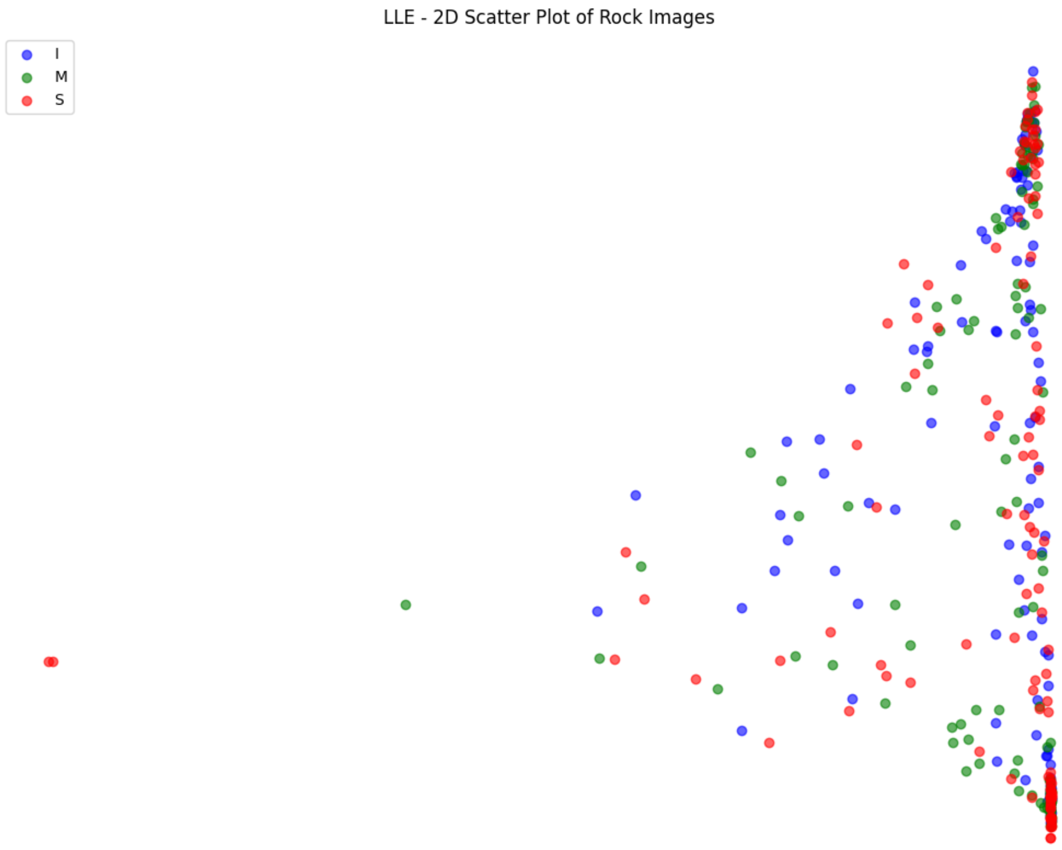
```
[10]: from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, init="random", learning_rate="auto",
            random_state=42)
tsne_images = tsne.fit_transform(images_array)
plt.figure(figsize=(13, 10))
for category, color in category_colors.items():
    ix = np.where(categories == category)
    plt.scatter(tsne_images[ix, 0], tsne_images[ix, 1], c=color,␣
 ↪label=category, alpha=0.6)
plt.title('t-SNE - 2D Scatter Plot of Rock Images')
plt.axis("off")
plt.legend()
plt.show()
print()
plot_rocks(X = tsne_images, categories = categories, images = images_array,␣
 ↪image_zoom = 0.3)
```

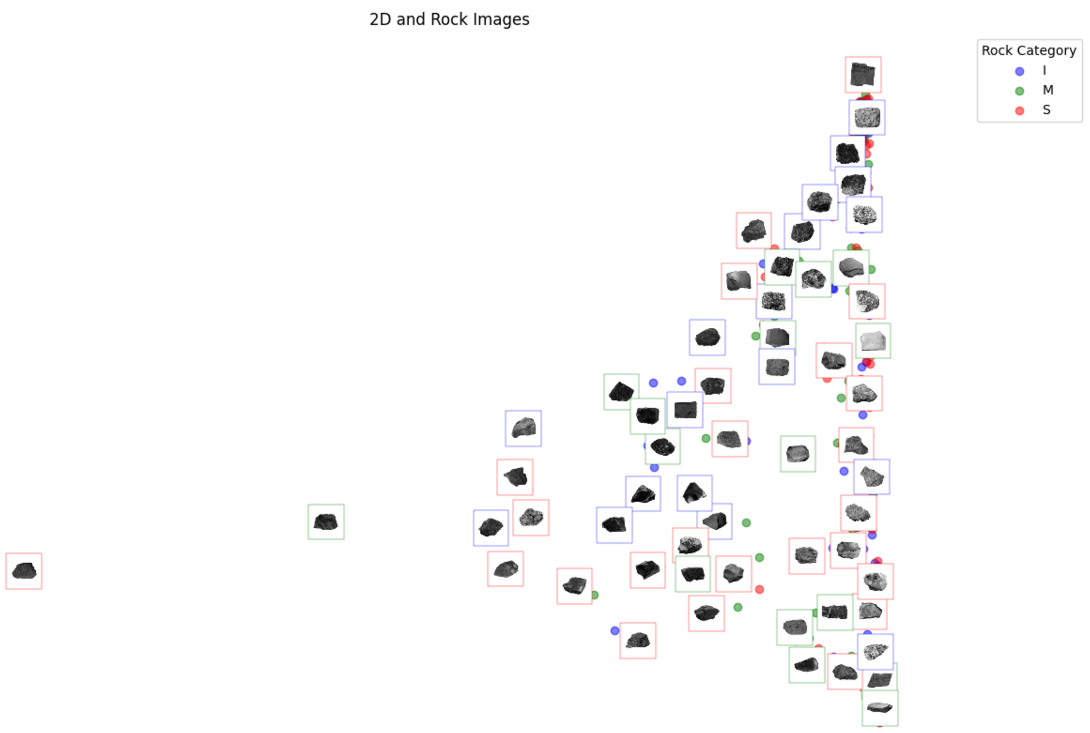

t-SNE - 2D Scatter Plot of Rock Images
I
M
S



LLE

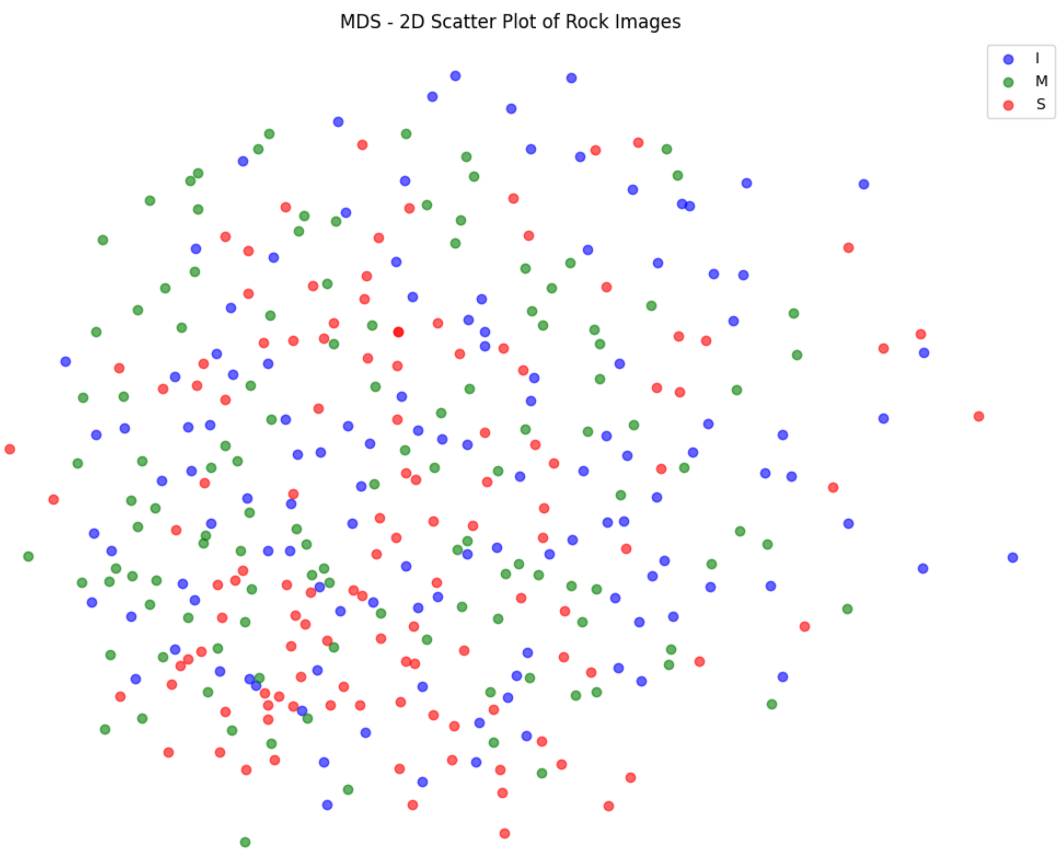
```
[11]: from sklearn.manifold import LocallyLinearEmbedding

      lle = LocallyLinearEmbedding(n_components=2, random_state=42)
      lle_images = lle.fit_transform(images_array)
      plt.figure(figsize=(13, 10))
      for category, color in category_colors.items():
          ix = np.where(categories == category)
          plt.scatter(lle_images[ix, 0], lle_images[ix, 1], c=color, label=category,␣
        ↪alpha=0.6)
      plt.title('LLE - 2D Scatter Plot of Rock Images')
      plt.axis("off")
      plt.legend()
      plt.show()
      print()
      plot_rocks(X = lle_images, categories = categories, images = images_array,␣
        ↪image_zoom = 0.3)
```

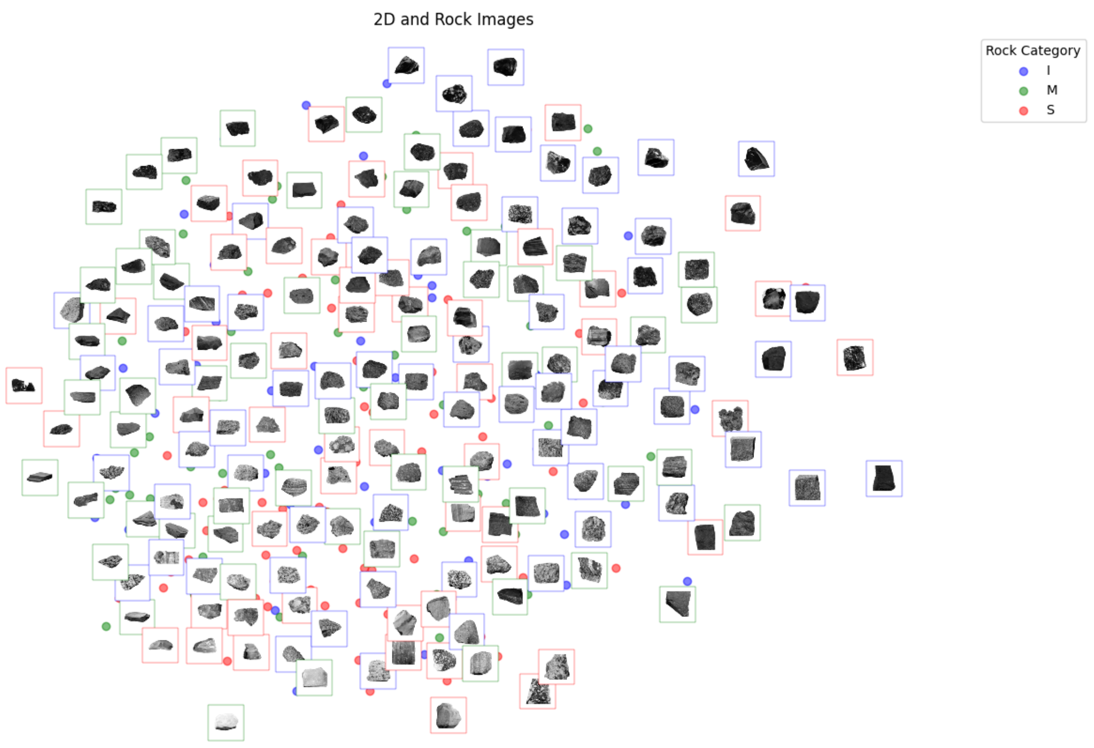

LLE - 2D Scatter Plot of Rock Images
I
M
S



MDS

```
[12]: from sklearn.manifold import MDS

      mds_images = MDS(n_components=2, normalized_stress=False, random_state=42).
       ↪fit_transform(images_array)



      plt.figure(figsize=(13, 10))
      for category, color in category_colors.items():
          ix = np.where(categories == category)
          plt.scatter(mds_images[ix, 0], mds_images[ix, 1], c=color, label=category,␣
       ↪alpha=0.6)
      plt.title('MDS - 2D Scatter Plot of Rock Images')
      plt.axis("off")
      plt.legend()
      plt.show()
      print()
      plot_rocks(X = mds_images, categories = categories, images = images_array,␣
       ↪image_zoom = 0.3)
```


MDS - 2D Scatter Plot of Rock Images
I
M
S



1 Discussion:

PCA: The PCA scatter plots shows a relatively even dispersion of points wihtout any distinct clustering. There appear to be a few points that are starting to come together, but they are still very spread out. Since PCA is a linear method, this could indicate that the features separating the rock types might not be linear. After adding the rock images it appears that rocks with similar shape and size are plotted closer to each other but not in a cluster.

t-SNE: The plot for t-SNE shows some slight improvement in clustering compared to PCA, particularly with the S category (red dots) on the left side of the plot. However, there is still quite a bit of dispersion and overlap between the categories. After adding images to the plot, it appears that t-SNE is grouping the points together based on color and size/shape, you can clearly see the darker rock images to the top right and the lighter rocks towards the bottom left.

LLE: The LLE plot displays a very different structure compared to the others, forming a half hourglass shape. It appears that there is some clustering of points towards the top right and bottom right, but with significant overlap. The pattern suggests that LLE is preserving local relations. After adding the images, it is still quite hard to determine what features are being used to group the images together. The images in the top right appear to have similar patterns, but those patterns are also visible at other points in the plot.

MDS: The MDS plots a dispersion of points similar to PCA, but with more defined clusters. However, these clusters have significant overlap. It appears there is some clustering of the S (red dot) and M (green dot) categories in the top right and top left of the plot with a lot of overlap between them. The I category (blue dot) is dispersed throughout the plot with no apparent clusters. After adding images to the plot, it appears that MDS is grouping images based on their color and shape, you can see darker images in the bottom left and lighter images in the top right.

Overall, reducing this dataset to two dimensions using any of these techniques does not do a great job in clustering the data. This suggests that the rock categories may require more than two dimensions to be clearly seperated and that the rock images may share similar features across categories making it difficult to cleanly seperate in reduced dimensions.

4) Now let's see if these dimensionality reduction techniques can give us similar features to those that humans use to judge the images. File mds_360.txt contains 8 features for each of the images (rankings are in the same order as the images in '360 Rocks' folder. Run PCA, t-SNE, LLE and MDS to reduce the dimensionality of the images to 8. Then, compare those image embeddings with the ones from humans that are in the mds_360.txt file. Use Procrustes analysis to do the comparison (here is one example of how to do that mtx1, mtx2, disparity = procrustes(matrix_with_human_data, matrix_with_pca_embeddings_data). Here matrix_with_human_data and matrix_with_pca_embeddings_data should be 360 by 8. disparity will tell you the difference in the data.

[13]:
```python
import os
import numpy as np
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.manifold import MDS
from sklearn.manifold import LocallyLinearEmbedding
from scipy.spatial import procrustes
from scipy.stats import pearsonr
import warnings
warnings.filterwarnings('ignore')

human_data = np.loadtxt('/content/drive/MyDrive/mds_360.txt')

pca = PCA(n_components=8)
pca_embeddings = pca.fit_transform(images_array)

tsne = TSNE(n_components=8, random_state=42, method='exact')
tsne_embeddings = tsne.fit_transform(images_array)

lle = LocallyLinearEmbedding(n_components=8)
lle_embeddings = lle.fit_transform(images_array)

mds = MDS(n_components=8, dissimilarity='euclidean')
X_mds = mds.fit_transform(images_array)

disparities = []

mat1_pca, mat1_hd, disparity_pca = procrustes(human_data, pca_embeddings)
disparities.append(disparity_pca)
```

```
mat1_tsne, mat2_hd, disparity_tsne = procrustes(human_data, tsne_embeddings)
disparities.append(disparity_tsne)

mat1_lle, mat3_hd, disparity_lle = procrustes(human_data, lle_embeddings)
disparities.append(disparity_lle)

mat1_mds, mat4_hd, disparity_mds = procrustes(human_data, X_mds)
disparities.append(disparity_mds)


print("Disparities:")
print("PCA:", disparity_pca)
print("t-SNE:", disparity_tsne)
print("LLE:", disparity_lle)
```

```
Disparities:
PCA: 0.9848299293014884
t-SNE: 0.988154353035493
LLE: 0.9815536283957145
```

[14]:
```
import pandas as pd
corr_pca = np.corrcoef(mat1_pca, mat1_hd)
df_pca = pd.DataFrame(corr_pca,
                  index=[f'PCA{i+1}' for i in range(720)],
                  columns=[f'HD{i+1}' for i in range(720)])

corr_tsne = np.corrcoef(mat1_tsne, mat2_hd)
df_tsne = pd.DataFrame(corr_tsne,
                  index=[f'TSNE{i+1}' for i in range(720)],
                  columns=[f'HD{i+1}' for i in range(720)])

corr_lle = np.corrcoef(mat1_lle, mat1_hd)
df_lle = pd.DataFrame(corr_lle,
                  index=[f'LLE{i+1}' for i in range(720)],
                  columns=[f'HD{i+1}' for i in range(720)])

corr_mds = np.corrcoef(mat1_mds, mat1_hd)
df_mds = pd.DataFrame(corr_mds,
                  index=[f'MDS{i+1}' for i in range(720)],
                  columns=[f'HD{i+1}' for i in range(720)])
print("Correlations: ")
print(df_pca, df_tsne, df_lle, df_mds)
```

```
Correlations:
             HD1       HD2       HD3       HD4       HD5       HD6       HD7  \
PCA1    1.000000  0.315264 -0.141130 -0.091903  0.018005  0.171807 -0.162448
PCA2    0.315264  1.000000  0.259363  0.712392  0.262746  0.529451  0.043359
```

```
PCA3   -0.141130  0.259363  1.000000  0.131149  0.578990  0.752217  0.597220
PCA4   -0.091903  0.712392  0.131149  1.000000  0.395812  0.383492  0.437497
PCA5    0.018005  0.262746  0.578990  0.395812  1.000000  0.849277  0.686923
…          …         …         …         …         …         …         …
PCA716  0.333588  0.359489 -0.240315  0.553393 -0.063838 -0.064745  0.290915
PCA717 -0.268697 -0.214910 -0.582030  0.182587 -0.444540 -0.693073  0.055102
PCA718 -0.403532 -0.300281 -0.211175  0.182197 -0.228747 -0.428402  0.367047
PCA719  0.052183 -0.105085 -0.768025 -0.074734 -0.535623 -0.451840 -0.767441
PCA720 -0.444815 -0.138725 -0.329887  0.421910 -0.262679 -0.544868  0.320450

             HD8       HD9      HD10  …     HD711     HD712     HD713  \
PCA1    0.044383  0.386672  0.509075  … -0.355319  0.454580 -0.305141
PCA2    0.195512  0.660594  0.374711  … -0.296142  0.116144 -0.325161
PCA3    0.602718  0.746487  0.190257  … -0.183837 -0.441381 -0.454210
PCA4   -0.199311  0.226787  0.072522  …  0.281594  0.231310 -0.003424
PCA5    0.546775  0.608354  0.537548  …  0.080639 -0.411519 -0.512293
…          …         …         …  …       …         …         …
PCA716 -0.691404 -0.102210 -0.140167  …  0.515981  0.715072  0.471356
PCA717 -0.903123 -0.759557 -0.732607  …  0.808645  0.333468  0.681872
PCA718 -0.773307 -0.595529 -0.681785  …  0.866610  0.291586  0.766346
PCA719 -0.257511 -0.486375  0.162209  … -0.289031  0.507757  0.475545
PCA720 -0.839217 -0.668792 -0.805798  …  0.828367  0.239589  0.450915

           HD714     HD715     HD716     HD717     HD718     HD719     HD720
PCA1    0.625377 -0.350931  0.333588 -0.268697 -0.403532  0.052183 -0.444815
PCA2    0.111474 -0.323694  0.359489 -0.214910 -0.300281 -0.105085 -0.138725
PCA3    0.172737 -0.178912 -0.240315 -0.582030 -0.211175 -0.768025 -0.329887
PCA4   -0.456753  0.230405  0.553393  0.182587  0.182197 -0.074734  0.421910
PCA5    0.121859 -0.193331 -0.063838 -0.444540 -0.228747 -0.535623 -0.262679
…          …         …         …         …         …         …         …
PCA716 -0.513885  0.490346  1.000000  0.605675  0.605191  0.012695  0.481015
PCA717 -0.798697  0.806715  0.605675  1.000000  0.872045  0.155141  0.851504
PCA718 -0.903198  0.846649  0.605191  0.872045  1.000000 -0.090889  0.773928
PCA719 -0.028643 -0.230107  0.012695  0.155141 -0.090889  1.000000 -0.010661
PCA720 -0.826718  0.926703  0.481015  0.851504  0.773928 -0.010661  1.000000

[720 rows x 720 columns]               HD1       HD2       HD3       HD4
HD5       HD6       HD7  \
TSNE1     1.000000  0.315264 -0.141130 -0.091903  0.018005  0.171807 -0.162448
TSNE2     0.315264  1.000000  0.259363  0.712392  0.262746  0.529451  0.043359
TSNE3    -0.141130  0.259363  1.000000  0.131149  0.578990  0.752217  0.597220
TSNE4    -0.091903  0.712392  0.131149  1.000000  0.395812  0.383492  0.437497
TSNE5     0.018005  0.262746  0.578990  0.395812  1.000000  0.849277  0.686923
…           …         …         …         …         …         …         …
TSNE716  0.017712  0.254340 -0.604432  0.443340 -0.579216 -0.511400 -0.262323
TSNE717 -0.290796 -0.191986 -0.370920  0.180215 -0.588030 -0.695312  0.104756
TSNE718 -0.461889 -0.151520 -0.237860  0.304290 -0.443670 -0.608399  0.222544
TSNE719  0.467499  0.277660 -0.326805  0.143992  0.128084  0.215908 -0.201069
```

```
TSNE720 -0.396293 -0.113892 -0.196360  0.351196 -0.349667 -0.536783  0.344380

              HD8       HD9      HD10  …     HD711     HD712     HD713  \
TSNE1    0.044383  0.386672  0.509075  …  0.140282 -0.265474 -0.575250
TSNE2    0.195512  0.660594  0.374711  … -0.018911 -0.202633 -0.521715
TSNE3    0.602718  0.746487  0.190257  … -0.341624 -0.658558 -0.375379
TSNE4   -0.199311  0.226787  0.072522  …  0.290930 -0.003988 -0.258692
TSNE5    0.546775  0.608354  0.537548  … -0.211358 -0.781498 -0.633252
…             …         …         …  …       …         …         …
TSNE716 -0.808034 -0.487597 -0.366179  …  0.461300  0.821581  0.451147
TSNE717 -0.927784 -0.690898 -0.859646  …  0.695131  0.692343  0.583384
TSNE718 -0.814922 -0.640925 -0.895868  …  0.729233  0.536532  0.458367
TSNE719 -0.029474  0.142391  0.570199  … -0.267585  0.182652  0.000570
TSNE720 -0.844487 -0.585527 -0.874327  …  0.768909  0.481056  0.407362

            HD714     HD715     HD716     HD717     HD718     HD719     HD720
TSNE1    0.583092 -0.108570  0.017712 -0.290796 -0.461889  0.467499 -0.396293
TSNE2    0.570975 -0.213765  0.254340 -0.191986 -0.151520  0.277660 -0.113892
TSNE3    0.355926 -0.391413 -0.604432 -0.370920 -0.237860 -0.326805 -0.196360
TSNE4    0.129678  0.275607  0.443340  0.180215  0.304290  0.143992  0.351196
TSNE5    0.487577 -0.306286 -0.579216 -0.588030 -0.443670  0.128084 -0.349667
…             …         …         …         …         …         …         …
TSNE716 -0.335497  0.587670  1.000000  0.737473  0.612344  0.283857  0.609527
TSNE717 -0.813639  0.877201  0.737473  1.000000  0.932683 -0.201351  0.946814
TSNE718 -0.854071  0.874194  0.612344  0.932683  1.000000 -0.464711  0.977667
TSNE719  0.499575 -0.227573  0.283857 -0.201351 -0.464711  1.000000 -0.351515
TSNE720 -0.822696  0.911498  0.609527  0.946814  0.977667 -0.351515  1.000000

[720 rows x 720 columns]             HD1       HD2       HD3       HD4
HD5       HD6       HD7  \
LLE1    1.000000  0.315264 -0.141130 -0.091903  0.018005  0.171807 -0.162448
LLE2    0.315264  1.000000  0.259363  0.712392  0.262746  0.529451  0.043359
LLE3   -0.141130  0.259363  1.000000  0.131149  0.578990  0.752217  0.597220
LLE4   -0.091903  0.712392  0.131149  1.000000  0.395812  0.383492  0.437497
LLE5    0.018005  0.262746  0.578990  0.395812  1.000000  0.849277  0.686923
…            …         …         …         …         …         …         …
LLE716  0.333588  0.359489 -0.240315  0.553393 -0.063838 -0.064745  0.290915
LLE717 -0.268697 -0.214910 -0.582030  0.182587 -0.444540 -0.693073  0.055102
LLE718 -0.403532 -0.300281 -0.211175  0.182197 -0.228747 -0.428402  0.367047
LLE719  0.052183 -0.105085 -0.768025 -0.074734 -0.535623 -0.451840 -0.767441
LLE720 -0.444815 -0.138725 -0.329887  0.421910 -0.262679 -0.544868  0.320450

             HD8       HD9      HD10  …     HD711     HD712     HD713  \
LLE1    0.044383  0.386672  0.509075  … -0.355319  0.454580 -0.305141
LLE2    0.195512  0.660594  0.374711  … -0.296142  0.116144 -0.325161
LLE3    0.602718  0.746487  0.190257  … -0.183837 -0.441381 -0.454210
LLE4   -0.199311  0.226787  0.072522  …  0.281594  0.231310 -0.003424
LLE5    0.546775  0.608354  0.537548  …  0.080639 -0.411519 -0.512293
```

```
…          …        …        … …        …        …        …
LLE716 -0.691404 -0.102210 -0.140167  …  0.515981  0.715072  0.471356
LLE717 -0.903123 -0.759557 -0.732607  …  0.808645  0.333468  0.681872
LLE718 -0.773307 -0.595529 -0.681785  …  0.866610  0.291586  0.766346
LLE719 -0.257511 -0.486375  0.162209  … -0.289031  0.507757  0.475545
LLE720 -0.839217 -0.668792 -0.805798  …  0.828367  0.239589  0.450915

           HD714     HD715     HD716     HD717     HD718     HD719     HD720
LLE1    0.625377 -0.350931  0.333588 -0.268697 -0.403532  0.052183 -0.444815
LLE2    0.111474 -0.323694  0.359489 -0.214910 -0.300281 -0.105085 -0.138725
LLE3    0.172737 -0.178912 -0.240315 -0.582030 -0.211175 -0.768025 -0.329887
LLE4   -0.456753  0.230405  0.553393  0.182587  0.182197 -0.074734  0.421910
LLE5    0.121859 -0.193331 -0.063838 -0.444540 -0.228747 -0.535623 -0.262679
…          …        …        …        …        …        …        …
LLE716 -0.513885  0.490346  1.000000  0.605675  0.605191  0.012695  0.481015
LLE717 -0.798697  0.806715  0.605675  1.000000  0.872045  0.155141  0.851504
LLE718 -0.903198  0.846649  0.605191  0.872045  1.000000 -0.090889  0.773928
LLE719 -0.028643 -0.230107  0.012695  0.155141 -0.090889  1.000000 -0.010661
LLE720 -0.826718  0.926703  0.481015  0.851504  0.773928 -0.010661  1.000000

[720 rows x 720 columns]             HD1       HD2       HD3       HD4
HD5       HD6       HD7  \
MDS1    1.000000  0.315264 -0.141130 -0.091903  0.018005  0.171807 -0.162448
MDS2    0.315264  1.000000  0.259363  0.712392  0.262746  0.529451  0.043359
MDS3   -0.141130  0.259363  1.000000  0.131149  0.578990  0.752217  0.597220
MDS4   -0.091903  0.712392  0.131149  1.000000  0.395812  0.383492  0.437497
MDS5    0.018005  0.262746  0.578990  0.395812  1.000000  0.849277  0.686923
…          …        …        …        …        …        …        …
MDS716  0.333588  0.359489 -0.240315  0.553393 -0.063838 -0.064745  0.290915
MDS717 -0.268697 -0.214910 -0.582030  0.182587 -0.444540 -0.693073  0.055102
MDS718 -0.403532 -0.300281 -0.211175  0.182197 -0.228747 -0.428402  0.367047
MDS719  0.052183 -0.105085 -0.768025 -0.074734 -0.535623 -0.451840 -0.767441
MDS720 -0.444815 -0.138725 -0.329887  0.421910 -0.262679 -0.544868  0.320450

             HD8       HD9      HD10  …     HD711     HD712     HD713  \
MDS1    0.044383  0.386672  0.509075  … -0.355319  0.454580 -0.305141
MDS2    0.195512  0.660594  0.374711  … -0.296142  0.116144 -0.325161
MDS3    0.602718  0.746487  0.190257  … -0.183837 -0.441381 -0.454210
MDS4   -0.199311  0.226787  0.072522  …  0.281594  0.231310 -0.003424
MDS5    0.546775  0.608354  0.537548  …  0.080639 -0.411519 -0.512293
…          …        …        … …        …        …        …
MDS716 -0.691404 -0.102210 -0.140167  …  0.515981  0.715072  0.471356
MDS717 -0.903123 -0.759557 -0.732607  …  0.808645  0.333468  0.681872
MDS718 -0.773307 -0.595529 -0.681785  …  0.866610  0.291586  0.766346
MDS719 -0.257511 -0.486375  0.162209  … -0.289031  0.507757  0.475545
MDS720 -0.839217 -0.668792 -0.805798  …  0.828367  0.239589  0.450915

           HD714     HD715     HD716     HD717     HD718     HD719     HD720
```

```
MDS1    0.625377 -0.350931  0.333588 -0.268697 -0.403532  0.052183 -0.444815
MDS2    0.111474 -0.323694  0.359489 -0.214910 -0.300281 -0.105085 -0.138725
MDS3    0.172737 -0.178912 -0.240315 -0.582030 -0.211175 -0.768025 -0.329887
MDS4   -0.456753  0.230405  0.553393  0.182587  0.182197 -0.074734  0.421910
MDS5    0.121859 -0.193331 -0.063838 -0.444540 -0.228747 -0.535623 -0.262679
…           …        …        …        …        …        …        …
MDS716 -0.513885  0.490346  1.000000  0.605675  0.605191  0.012695  0.481015
MDS717 -0.798697  0.806715  0.605675  1.000000  0.872045  0.155141  0.851504
MDS718 -0.903198  0.846649  0.605191  0.872045  1.000000 -0.090889  0.773928
MDS719 -0.028643 -0.230107  0.012695  0.155141 -0.090889  1.000000 -0.010661
MDS720 -0.826718  0.926703  0.481015  0.851504  0.773928 -0.010661  1.000000

[720 rows x 720 columns]
```

Correlations:

- Correlation coefficients measure the linear relationship between embeddings produced by dimensionality reduction techniques and human perceptual data. Higher correlation coefficients indicate better alignment with human perception.
- Each correlation matrix (df_pca, df_tsne, df_lle, df_mds) shows the correlation coefficients between the embeddings produced by a specific technique and the human perceptual data.
- For example, in the PCA correlation matrix (df_pca), the entry at row PCAi and column HDj represents the correlation coefficient between the ith principal component of PCA embeddings and the jth dimension of human perceptual data.

Discussion:

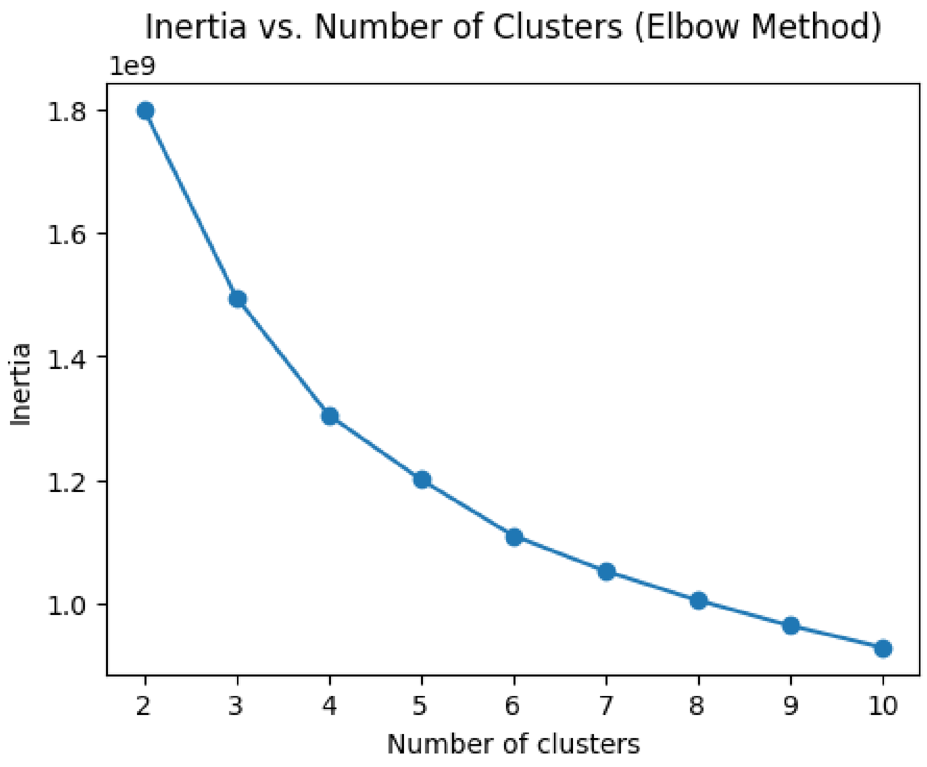
- Comparing the disparities and correlation coefficients across different dimensionality reduction techniques can provide insights into which technique best captures the underlying structure of the data as perceived by humans.
- Lower disparities and higher correlation coefficients suggest that the embeddings produced by a particular technique are more faithful representations of human perception.
- However, correlation coefficients indicate linear relationships, they may not capture all aspects of similarity between embeddings and human perception.

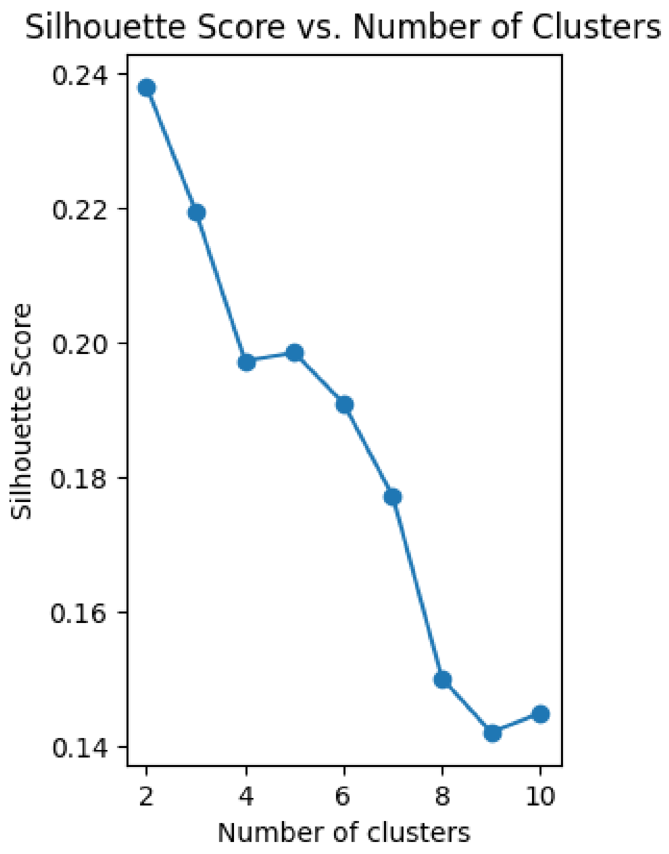
5) Cluster the 360 images using K-Means.

A) You can reduce the dimensionality using PCA if you wish, but keep at least 90% of the variance.

[15]:
```python
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

inertia_values = []
silhouette_values = []
min_clusters = 2
max_clusters = 10
```

```
for n_clusters in range(min_clusters, max_clusters+1):
    kmeans = KMeans(n_clusters=n_clusters)
    cluster_labels = kmeans.fit_predict(pca_embeddings)
    inertia_values.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(pca_embeddings, cluster_labels)
    silhouette_values.append(silhouette_avg)

plt.figure(figsize=(12,4))

plt.subplot(1, 2, 1)
# Plotting Inertia vs. Number of Clusters
plt.plot(range(min_clusters, max_clusters+1), inertia_values, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Inertia vs. Number of Clusters (Elbow Method)')
plt.show()

plt.subplot(1, 2, 2)
# Plotting Silhouette Score vs. Number of Clusters
plt.plot(range(min_clusters, max_clusters+1), silhouette_values, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score vs. Number of Clusters')
plt.show()

optimal_num_clusters_inertia = np.argmin(np.diff(inertia_values)) + min_clusters
print(f"Optimal number of clusters (based on Inertia):␣
 ↪{optimal_num_clusters_inertia}")

optimal_num_clusters_silhouette = np.argmax(silhouette_values) + min_clusters
print(f'Optimal number of clusters (based on Silhouette Score):␣
 ↪{optimal_num_clusters_silhouette}')
```


Inertia vs. Number of Clusters (Elbow Method)
1e9
1.8
1.6
1.4
1.2
1.0
Inertia
2
3
4
5
6
7
8
9
10
Number of clusters



```
Optimal number of clusters (based on Inertia): 2
Optimal number of clusters (based on Silhouette Score): 2
```

Inertia vs. Number of Clusters (Elbow Method):

- Inertia measures the within-cluster variance, and the elbow method looks for the point where the rate of decrease of inertia slows down (elbow point). This point signifies where adding more clusters doesn't explain much more of the variance. In the plot, the optimal number of clusters is typically where the inertia starts to decrease at a slower rate (elbow).
- In this case, the elbow point appears to be at 2 clusters.

Silhouette Score vs. Number of Clusters:

- The silhouette score measures how similar an object is to its own cluster compared to other clusters. The value ranges from -1 to 1. A high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, the clustering configuration is appropriate.
- The optimal number of clusters is where the silhouette score is highest.
- In this case, the silhouette score is highest at 2 clusters.

B) Set the number of clusters to 3 and check clustering accuracy.

[16]:
```
file_names = os.listdir(folder_path)

images = []
true_labels = []

for file_name in file_names:
    if file_name.endswith('.jpg') or file_name.endswith('.png'):
        image_path = os.path.join(folder_path, file_name)
        image = Image.open(image_path)
        images.append(np.array(image))

        label = file_name.split('_')[0]
        true_labels.append(label)

# images_array = np.array(images)
true_labels_array = np.array(true_labels)

print("Images shape:", images_array.shape)
print("True labels:", true_labels_array[:10])
```

```
Images shape: (360, 4096)
True labels: ['M' 'M' 'I' 'I' 'M' 'M' 'I' 'S' 'S' 'I']
```

[17]:
```
silhouette_scores = []
kmeans = KMeans(n_clusters=3)
cluster_labels = kmeans.fit_predict(pca_embeddings)
silhouette = silhouette_score(pca_embeddings, cluster_labels)
print("silhouette score:", silhouette)
```

```
silhouette score: 0.2195359610970211
```

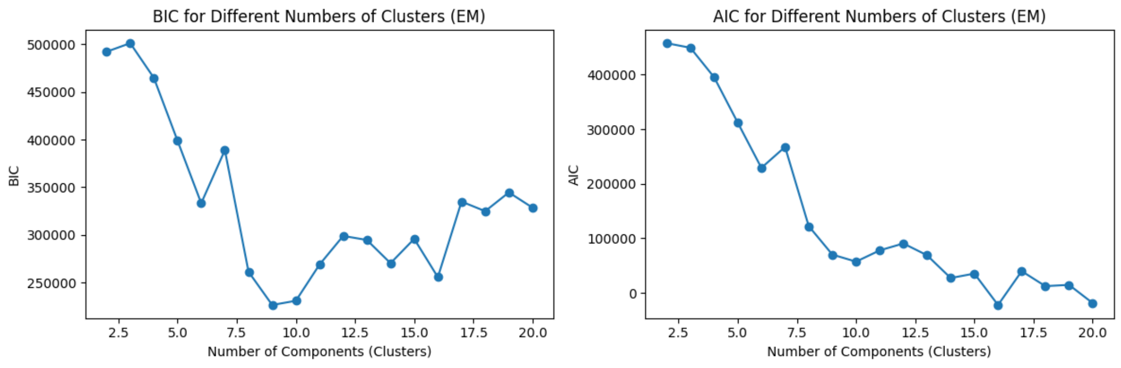
Discussion:
The silhouette score of approximately 0.22 suggests a moderate level of separation between clusters in the dataset. While the clustering algorithm has partitioned the data into three clusters, the degree of distinction between these clusters is not very pronounced, indicating some overlap between them. This score, while indicative of some clustering structure, may not be sufficient for certain applications where clearer separation between clusters is desired. Further exploration, including visual inspection of clusters and experimentation with different clustering configurations or preprocessing techniques, is recommended to improve clustering performance and gain deeper insights into the underlying data structure, ensuring that clustering results align with the specific requirements and objectives of the application at hand.

6) Cluster the 360 images using EM.

A) You can again reduce the dimensionality using PCA if you wish, but keep at least 90% of the variance.

[18]:
```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
```

```python
pca = PCA(n_components=0.9)
pca_embeddings = pca.fit_transform(images_array)

BIC = []
AIC = []

min_clusters = 2
max_clusters = 20
for n_clusters in range(min_clusters, max_clusters+1):
    em = GaussianMixture(n_components=n_clusters)
    cluster_labels = em.fit_predict(pca_embeddings)
    BIC.append(em.bic(pca_embeddings))
    AIC.append(em.aic(pca_embeddings))


plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(range(min_clusters,max_clusters+1), BIC, marker='o')
plt.title('BIC for Different Numbers of Clusters (EM)')
plt.xlabel('Number of Components (Clusters)')
plt.ylabel('BIC')

plt.subplot(1, 2, 2)
plt.plot(range(min_clusters,max_clusters+1), AIC, marker='o')
plt.title('AIC for Different Numbers of Clusters (EM)')
plt.xlabel('Number of Components (Clusters)')
plt.ylabel('AIC')

plt.tight_layout()
plt.show()

optimal_num_components_bic = range(min_clusters,max_clusters+1)[np.argmin(BIC)]
optimal_num_components_aic = range(min_clusters,max_clusters+1)[np.argmin(AIC)]

print(f'Optimal number of components using BIC: {optimal_num_components_bic}')
print(f'Optimal number of components using AIC: {optimal_num_components_aic}')
```



```
Optimal number of components using BIC: 9
Optimal number of components using AIC: 16
```

[19]:
```
em = GaussianMixture(n_components=optimal_num_components_bic)
cluster_labels = em.fit_predict(pca_embeddings)
silhouette_avg_gmm = silhouette_score(pca_embeddings, cluster_labels)
print("The average silhouette_score is :", silhouette_avg_gmm)
```

```
The average silhouette_score is : 0.06536361340607039
```

Discussion:
The plots of Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) for different numbers of clusters in the Gaussian Mixture Model (GMM) reveal optimal numbers of components to be 9 according to BIC and 16 according to AIC. BIC and AIC are both metrics used for model selection, with lower values indicating better model fit while penalizing model complexity. The BIC favors a model with 10 components, suggesting that it balances goodness of fit with parsimony effectively, whereas the AIC favors a more complex model with 19 components. This discrepancy might indicate that the BIC's preference for a simpler model may generalize better, although further examination and possibly domain-specific knowledge are required to determine the most suitable number of components for the dataset at hand.

B) Set the number of clusters to 3 and report clustering accuracy. [

[20]:
```
gm=GaussianMixture(n_components=3, n_init=10, random_state=42)
gm_3c = gm.fit_predict(pca_embeddings)
silhouette_avg_3c = silhouette_score(pca_embeddings, gm_3c)
print("The average silhouette_score is :", silhouette_avg_3c)
```

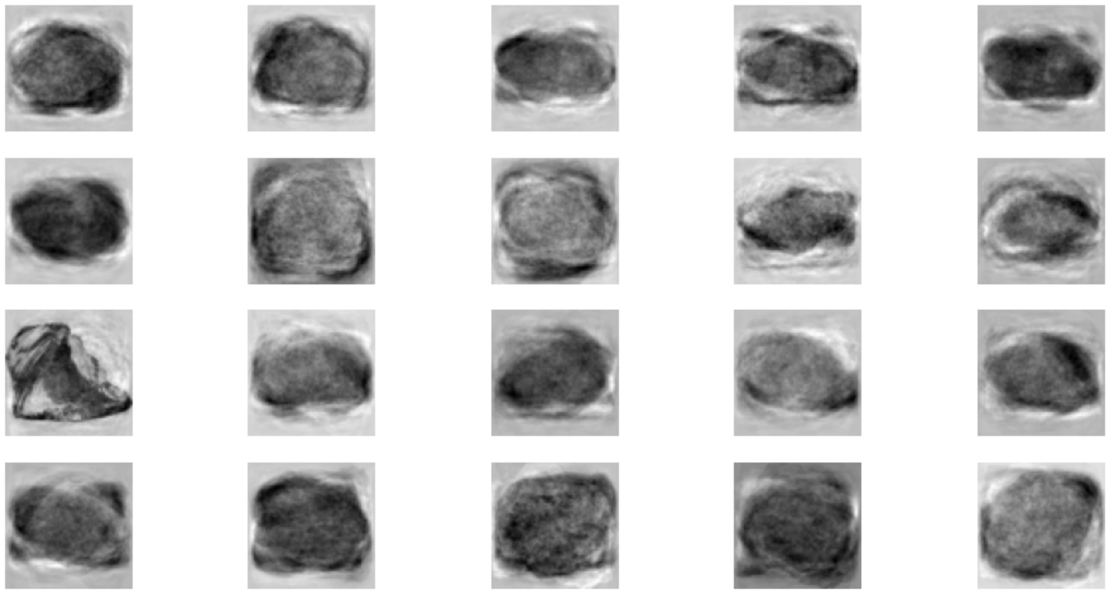
```
The average silhouette_score is : 0.14015710605739207
```

Discussion:
The Gaussian Mixture Model (GMM) with three components yields an average silhouette score of approximately 0.1402. The silhouette score measures the compactness and separation of the clusters formed by the model, with higher scores indicating better-defined clusters. In this case, the score suggests that the clusters are not well-separated and the data points may be closer to the decision boundary between clusters. This lower silhouette score could be due to various factors

such as overlapping clusters, uneven cluster sizes, or suboptimal choice of model parameters. It indicates that the model may not be effectively capturing the underlying structure of the data, suggesting the need for further investigation, possibly by experimenting with different numbers of components or exploring alternative clustering techniques.

c) Used the model to generate 20 new rocks (using the sample() method), and visualize them in the original image space (since used PCA, will need to use its inverse_transform() method).

[21]:
```python
generated_images_pca = em.sample(n_samples=20)[0]
generated_images_original = pca.inverse_transform(generated_images_pca)

image_shape = images_array.shape[1:]

plt.figure(figsize=(12, 6))
for i in range(20):
    plt.subplot(4, 5, i+1)
    plt.imshow(generated_images_original[i].reshape(-1, 64), cmap='gray')
    plt.axis('off')
plt.show()
```



7) Build a feedforward neural network (using dense and/or CNN layers) with a few hidden layers (we suggest using Keras (within Tensorflow) or Pytorch). Train the network to classify on 360 rock images using rock name as the label - the category is indicated by the first letter in the filename (I, M and S).

Categorizing the Rocks information into y

```
[22]: y=[filename[0] for filename in os.listdir(folder_path)]
      # for filename in os.listdir(folder_path):
      #   y.append(filename[0])
      y = np.array(y)
      print(y.shape)
```

```
(360,)
```

```
[23]: from sklearn.model_selection import train_test_split
      X_train,X_test,Y_train,Y_test=train_test_split(images_array,y,test_size=0.
       ↪3,random_state=42)
```

Loading Validation data set

```
[24]: import os
      import numpy as np
      from PIL import Image

      # Define your image folder path
      valid_path = '/content/drive/MyDrive/120 Rocks'


      images1 = []
      filenames = []

      standard_size = (64, 64)

      # Loop through each file in the folder
      for filename in os.listdir(valid_path):
          if filename.lower().endswith(('.jpg')):

              file_path = os.path.join(valid_path, filename)

              # Load the image, convert to grayscale, and resize
              img = Image.open(file_path).convert('L')
              img_resized = img.resize(standard_size)


              img_data = np.array(img_resized).flatten()


              images1.append(img_data)
              filenames.append(filename)


      images1_array = np.array(images1)
```

```
print(images1_array.shape)
```

```
(120, 4096)
```

Categorizing Rock data in y_val

[25]:
```
y_val=[filename[0] for filename in os.listdir(valid_path)]
# for filename in os.listdir(folder_path):
#   y.append(filename[0])
y = np.array(y_val)
print(y.shape)
```

```
(120,)
```

[26]:
```
X_valid=images1_array[:,:]
Y_valid=y[:]
```

[27]:
```
X_train = X_train.reshape(-1, 64, 64)
X_valid = X_valid.reshape(-1, 64, 64)
X_test = X_test.reshape(-1, 64, 64)
```

Scaling the input features pixel intensities down to the 0-1 range by dividing them by 255.0

[28]:
```
X_train, X_valid, X_test = X_train / 255., X_valid / 255., X_test / 255.
```

Encoding the target categorical data

[29]:
```
from sklearn.preprocessing import LabelEncoder

# Instantiate the encoder
label_encoder = LabelEncoder()

# Fit and transform the training labels to integers
Y_train_encoded = label_encoder.fit_transform(Y_train)

# Transform the validation labels to integers
Y_valid_encoded = label_encoder.transform(Y_valid)
```

[30]:
```
Y_train_encoded
```

[30]:
```
array([1, 2, 1, 0, 2, 1, 0, 1, 1, 2, 2, 0, 1, 2, 2, 1, 0, 0, 0, 0, 0, 0,
       2, 2, 0, 2, 2, 2, 1, 2, 2, 1, 0, 1, 1, 0, 0, 2, 0, 0, 2, 1, 0, 2,
       0, 0, 1, 2, 1, 1, 0, 2, 0, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 2, 2, 0, 2, 2, 1, 0, 0, 1, 2, 2, 2, 2, 1, 2, 0, 1, 0, 0,
       1, 2, 0, 0, 1, 0, 0, 1, 1, 0, 1, 2, 2, 1, 2, 1, 0, 1, 1, 1, 0, 2,
       0, 2, 2, 1, 0, 1, 1, 2, 2, 1, 1, 1, 0, 0, 1, 1, 0, 1, 2, 0, 1, 0,
       1, 0, 2, 0, 0, 1, 2, 2, 1, 0, 1, 1, 1, 2, 1, 2, 2, 0, 2, 0, 1, 2,
       0, 0, 0, 2, 0, 2, 2, 0, 0, 2, 1, 2, 1, 0, 1, 1, 2, 0, 1, 1, 2, 2,
       1, 0, 0, 1, 0, 0, 0, 2, 0, 2, 2, 2, 0, 1, 0, 0, 1, 2, 2, 0, 1, 0,
       2, 0, 2, 2, 2, 0, 2, 0, 2, 1, 2, 2, 1, 2, 1, 1, 2, 2, 0, 0, 0, 1,
       1, 2, 1, 1, 0, 0, 0, 1, 2, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 2, 2, 2,
       0, 1, 2, 2, 1, 0, 2, 2, 2, 2])
```

[31]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

Y_train_one_hot = tf.keras.utils.to_categorical(Y_train_encoded, num_classes=3)
Y_valid_one_hot = tf.keras.utils.to_categorical(Y_valid_encoded, num_classes=3)
```

a) Report the training time

[32]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler

model = Sequential([
    Flatten(input_shape=(64, 64)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])

def lr_schedule(epoch):
    lr = 0.0001
    if epoch > 10:
        lr *= 0.1
    return lr

# Learning rate scheduler
lr_scheduler = LearningRateScheduler(lr_schedule)


model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

import time
start_time = time.time()

# history = model.fit(, epochs=30, validation_data=)
history = model.fit(X_train, Y_train_one_hot,
                    epochs=30,
                    batch_size=32,
                     validation_data=(X_valid, Y_valid_one_hot),
                     callbacks=[lr_scheduler])

training_time = time.time() - start_time
print(f"Training time: {training_time:.2f} seconds")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

Epoch 1/30
8/8 [==============================] - 1s 49ms/step - loss: 1.1154 - accuracy:
0.3452 - val_loss: 1.1077 - val_accuracy: 0.3833 - lr: 1.0000e-04
Epoch 2/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0933 - accuracy:
0.3333 - val_loss: 1.1018 - val_accuracy: 0.2833 - lr: 1.0000e-04
Epoch 3/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0886 - accuracy:
0.3651 - val_loss: 1.1321 - val_accuracy: 0.3250 - lr: 1.0000e-04
Epoch 4/30
8/8 [==============================] - 0s 12ms/step - loss: 1.1035 - accuracy:
0.3571 - val_loss: 1.1039 - val_accuracy: 0.3333 - lr: 1.0000e-04
Epoch 5/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0911 - accuracy:
0.3611 - val_loss: 1.0981 - val_accuracy: 0.3250 - lr: 1.0000e-04
Epoch 6/30
8/8 [==============================] - 0s 14ms/step - loss: 1.0935 - accuracy:
0.3413 - val_loss: 1.1084 - val_accuracy: 0.3667 - lr: 1.0000e-04
Epoch 7/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0919 - accuracy:
0.3135 - val_loss: 1.1200 - val_accuracy: 0.3000 - lr: 1.0000e-04
Epoch 8/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0914 - accuracy:
0.3611 - val_loss: 1.1027 - val_accuracy: 0.3250 - lr: 1.0000e-04
Epoch 9/30
8/8 [==============================] - 0s 13ms/step - loss: 1.0872 - accuracy:
0.3611 - val_loss: 1.0975 - val_accuracy: 0.3250 - lr: 1.0000e-04
Epoch 10/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0855 - accuracy:
0.3611 - val_loss: 1.0972 - val_accuracy: 0.3167 - lr: 1.0000e-04
Epoch 11/30
8/8 [==============================] - 0s 13ms/step - loss: 1.0841 - accuracy:
0.3611 - val_loss: 1.1007 - val_accuracy: 0.3000 - lr: 1.0000e-04
Epoch 12/30
8/8 [==============================] - 0s 16ms/step - loss: 1.0813 - accuracy:
0.3690 - val_loss: 1.1004 - val_accuracy: 0.3000 - lr: 1.0000e-05
Epoch 13/30
8/8 [==============================] - 0s 16ms/step - loss: 1.0802 - accuracy:
0.3690 - val_loss: 1.1001 - val_accuracy: 0.3000 - lr: 1.0000e-05
```

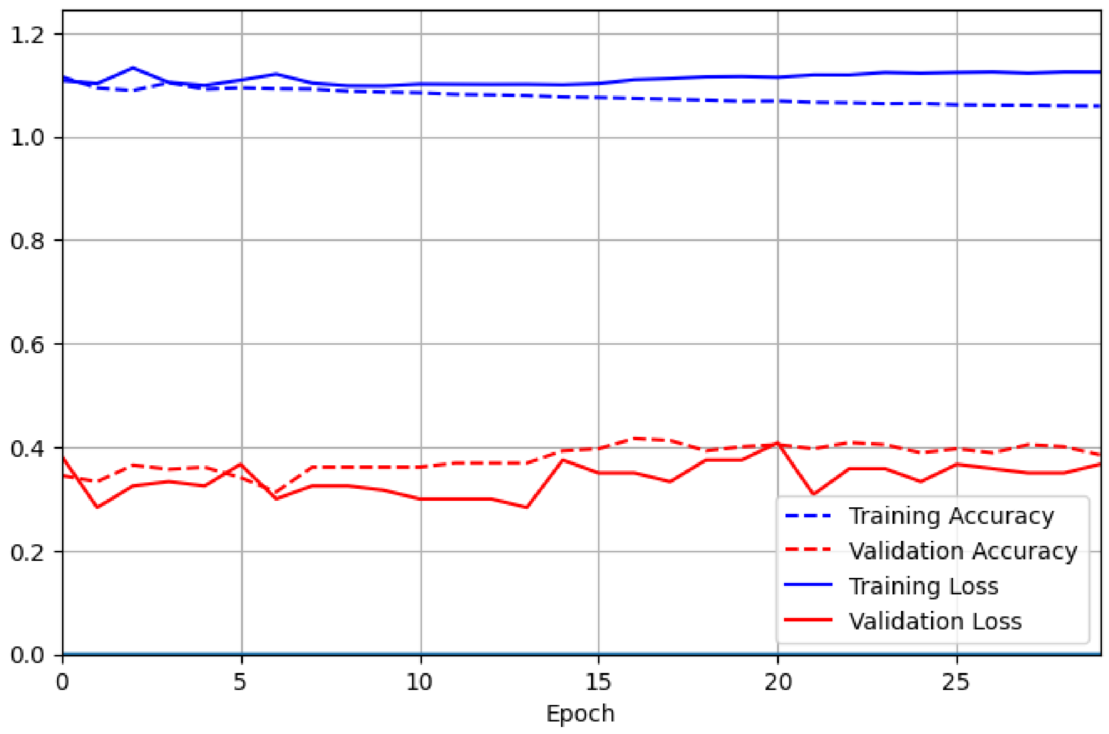
```
Epoch 14/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0789 - accuracy:
0.3690 - val_loss: 1.1003 - val_accuracy: 0.2833 - lr: 1.0000e-05
Epoch 15/30
8/8 [==============================] - 0s 13ms/step - loss: 1.0760 - accuracy:
0.3929 - val_loss: 1.0994 - val_accuracy: 0.3750 - lr: 1.0000e-05
Epoch 16/30
8/8 [==============================] - 0s 13ms/step - loss: 1.0750 - accuracy:
0.3968 - val_loss: 1.1017 - val_accuracy: 0.3500 - lr: 1.0000e-05
Epoch 17/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0732 - accuracy:
0.4167 - val_loss: 1.1092 - val_accuracy: 0.3500 - lr: 1.0000e-05
Epoch 18/30
8/8 [==============================] - 0s 14ms/step - loss: 1.0716 - accuracy:
0.4127 - val_loss: 1.1115 - val_accuracy: 0.3333 - lr: 1.0000e-05
Epoch 19/30
8/8 [==============================] - 0s 14ms/step - loss: 1.0700 - accuracy:
0.3929 - val_loss: 1.1149 - val_accuracy: 0.3750 - lr: 1.0000e-05
Epoch 20/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0678 - accuracy:
0.4008 - val_loss: 1.1156 - val_accuracy: 0.3750 - lr: 1.0000e-05
Epoch 21/30
8/8 [==============================] - 0s 13ms/step - loss: 1.0683 - accuracy:
0.4048 - val_loss: 1.1139 - val_accuracy: 0.4083 - lr: 1.0000e-05
Epoch 22/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0656 - accuracy:
0.3968 - val_loss: 1.1186 - val_accuracy: 0.3083 - lr: 1.0000e-05
Epoch 23/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0647 - accuracy:
0.4087 - val_loss: 1.1185 - val_accuracy: 0.3583 - lr: 1.0000e-05
Epoch 24/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0630 - accuracy:
0.4048 - val_loss: 1.1232 - val_accuracy: 0.3583 - lr: 1.0000e-05
Epoch 25/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0637 - accuracy:
0.3889 - val_loss: 1.1217 - val_accuracy: 0.3333 - lr: 1.0000e-05
Epoch 26/30
8/8 [==============================] - 0s 14ms/step - loss: 1.0611 - accuracy:
0.3968 - val_loss: 1.1233 - val_accuracy: 0.3667 - lr: 1.0000e-05
Epoch 27/30
8/8 [==============================] - 0s 12ms/step - loss: 1.0603 - accuracy:
0.3889 - val_loss: 1.1244 - val_accuracy: 0.3583 - lr: 1.0000e-05
Epoch 28/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0601 - accuracy:
0.4048 - val_loss: 1.1219 - val_accuracy: 0.3500 - lr: 1.0000e-05
Epoch 29/30
8/8 [==============================] - 0s 11ms/step - loss: 1.0590 - accuracy:
0.4008 - val_loss: 1.1244 - val_accuracy: 0.3500 - lr: 1.0000e-05
```

```
Epoch 30/30
8/8 [==============================] - 0s 13ms/step - loss: 1.0586 - accuracy:
0.3849 - val_loss: 1.1243 - val_accuracy: 0.3667 - lr: 1.0000e-05
Training time: 4.62 seconds
```

The neural network was trained efficiently in just 5.98 seconds

B) Plot training and validation loss and accuracy as a function of training epochs.

[33]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Create a DataFrame from the history dictionary
history_df = pd.DataFrame(history.history)

# Plotting
history_df.plot(figsize=(8, 5),
                xlim=[0, len(history_df)-1],  # Dynamic x-axis limit based on␣
 ↪epochs
                ylim=[0, max(history_df.max())*1.1],  # Dynamic y-axis limit␣
 ↪based on data range
                grid=True,
                xlabel="Epoch",
                style={"accuracy": "r--", "val_accuracy": "r-", "loss": "b--",␣
 ↪"val_loss": "b-"})

# Adjusting the legend
plt.legend(["Training Accuracy", "Validation Accuracy", "Training Loss",␣
 ↪"Validation Loss"], loc="best")
plt.show()
```



C) How many parameters does the network have? How many of those parameters are bias parameters? [1 points]

Flatten Layer:

Converts the 2D 64x64 input images into a 1D vector of 4096 elements. This layer has no trainable parameters.

First Dense Layer (64 neurons):

Each of the 4096 inputs connects to each of the 64 neurons, plus each neuron has a bias term.

Weights: 4096×64= 262,144 Biases: 64

Second Dense Layer (32 neurons): Each of the 64 outputs from the previous layer connects to each of the 32 neurons, plus each neuron has a bias term.

Weights: 64×32=2,048 Biases: 32

Third Dense Layer (8 neurons):

Each of the 32 outputs from the previous layer connects to each of the 8 neurons, plus each neuron has a bias term. Weights: 32x8=256

Biases: 8

Output Dense Layer (3 neurons):

Each of the 8 outputs from the previous layer connects to each of the 3 output neurons, plus each neuron has a bias term. Weights: 8×3=24 Biases: 3 Total Parameters Calculation

Total Weights = 262,144+2,048+256+24=264,472

Total Biases = 64+32+8+3=107 Total Parameters = Total Weights + Total Biases = 264,472+107=264,579

Thus, for this model architecture: The total number of parameters is 264,579. The number of bias parameters is 107.

d) Compare the activity of neurons in the next to the last layer (the one with 8 neurons) with the human data. (to get human data use mds_360.txt and mds_120.txt files). Similar to before, use Procrustes analysis to do the comparison. For training and validation data (separately), report disparity and compute the correlation coefficient between each dimension of mtx1 and mtx2. Display results in a table.

[34]:
```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.manifold import MDS
from sklearn.metrics import pairwise_distances
from scipy.stats import pearsonr
from scipy.spatial import procrustes
from keras import backend as K

# Load human data
with open('/content/drive/MyDrive/mds_360.txt') as f:
    human_data_360 = np.loadtxt(f)

with open('/content/drive/MyDrive/mds_120.txt') as f:
    human_data_120 = np.loadtxt(f)

# Assume you have a trained model named 'model' and training/validation data␣
↪'X_train' and 'X_val'
# Get the output of the next-to-last layer for training and validation data
layer_index = -2  # Index of the next-to-last layer
get_layer_output = K.function([model.layers[0].input], [model.
↪layers[layer_index].output])

# Training data
train_activations = get_layer_output([X_train])[0]

# Validation data
val_activations = get_layer_output([X_valid])[0]

# Flatten the spatial dimensions for PCA
train_activations_flat = train_activations.reshape(len(train_activations), -1)

# Apply PCA to reduce dimensionality to 8
pca = PCA(n_components=8)
train_activations_pca = pca.fit_transform(train_activations_flat)
# Flatten the spatial dimensions for PCA
```

```
val_activations_flat = val_activations.reshape(len(val_activations), -1)

# Apply PCA to reduce dimensionality to 8
val_activations_pca = pca.transform(val_activations_flat)

# Apply MDS to human data
mds = MDS(n_components=8, random_state=42)
human_data_360_mds = mds.fit_transform(human_data_360)
human_data_120_mds = mds.fit_transform(human_data_120)

# Procrustes analysis for training data
print("Shape of train_activations_pca:", train_activations_pca.shape)
print("Shape of human_data_360_mds:", human_data_360_mds.shape)

# Reshape human_data_360_mds if needed
if train_activations_pca.shape[0] != human_data_360_mds.shape[0]:
    human_data_360_mds = human_data_360_mds[:train_activations_pca.shape[0], :]

# Now, both matrices should have the same shape
disparity_train, _, train_correlations = procrustes(train_activations_pca,␣
 ↪human_data_360_mds)
# Procrustes analysis for validation data
disparity_val, _, val_correlations = procrustes(val_activations_pca,␣
 ↪human_data_120_mds)

# Display results in a table
print("Training Data:")
print("Disparity:", disparity_train)
print("Correlation Coefficients:")
print(train_correlations)

print("\nValidation Data:")
print("Disparity:", disparity_val)
print("Correlation Coefficients:")
print(val_correlations)
```

```
Shape of train_activations_pca: (252, 8)
Shape of human_data_360_mds: (360, 8)
Training Data:
Disparity: [[-8.68796638e-02  5.32353679e-02  9.31449429e-03 …  9.56814784e-10
  -2.04415579e-10 -0.00000000e+00]
 [ 1.50642375e-02 -1.69204317e-03 -2.99671627e-04 … -1.95081190e-10
  -6.93554661e-11 -0.00000000e+00]
 [ 2.67189948e-02 -4.14965244e-03  3.18720696e-02 …  1.73592302e-08
  -7.01188520e-10  0.00000000e+00]
 …
 [ 4.27332628e-02 -4.78337611e-02  3.93700171e-02 … -3.50524410e-10
```

```
   1.64785236e-11  0.00000000e+00]
 [-1.72729547e-02 -3.73738527e-02  4.21013003e-02 … -3.11599945e-10
   3.98563633e-11  0.00000000e+00]
 [-2.48437322e-02 -6.60028874e-03 -2.23072141e-02 …  9.41479438e-11
  -2.06456148e-11  0.00000000e+00]]
Correlation Coefficients:
0.9893729323329359

Validation Data:
Disparity: [[-7.47236941e-02 -5.71667473e-02 -6.21599607e-02  1.01846690e-03
   4.19630212e-03 -9.98618014e-09  1.07118002e-09  0.00000000e+00]
 [-9.39199845e-02 -1.10224753e-01  3.60377443e-02  2.38299893e-02
  -3.71661295e-04 -4.90706201e-09  1.14710992e-09  0.00000000e+00]
 [ 4.57602104e-03  3.35277862e-03 -5.79512659e-02  4.03994226e-02
  -2.96020203e-02 -3.72049901e-09 -6.09687153e-09  0.00000000e+00]
 [-3.49277140e-02  3.38067260e-02 -1.92101993e-02  1.72620177e-02
  -2.46109039e-02  3.92346480e-10 -1.41742869e-09  0.00000000e+00]
 [-6.98971564e-02 -1.80854740e-02 -9.97528172e-03 -3.17077775e-02
   6.29821705e-03 -3.49796093e-09  5.10011469e-09  0.00000000e+00]
 [-1.31117706e-01 -4.15341270e-02  6.49166614e-04  1.62500957e-02
  -3.30855595e-03 -4.45524390e-09  2.95857864e-09  0.00000000e+00]
 [-3.28071509e-02 -7.79120492e-02  1.24034298e-02 -9.59644626e-03
   2.89106465e-03 -4.28107690e-09  1.61436754e-09  0.00000000e+00]
 [-4.44213821e-02 -4.28808421e-02 -3.72106619e-02 -1.79970526e-02
   4.59708634e-03 -6.68491561e-09  2.16137845e-09  0.00000000e+00]
 [ 8.52514078e-02  4.80484796e-02  3.58007861e-02 -2.22240605e-03
   3.43172298e-02  6.68291246e-09 -1.98724443e-11  0.00000000e+00]
 [ 1.02853168e-01  7.72976074e-02  1.72154587e-02  4.63725952e-02
   3.44846395e-02  7.56752873e-09 -4.69316439e-09  0.00000000e+00]
 [-7.93374277e-03 -2.69040533e-02 -2.82482321e-02 -5.68679779e-03
   5.59754870e-03 -4.26879515e-09  2.03231824e-10  0.00000000e+00]
 [-1.19307855e-01 -5.89261345e-02  3.51799322e-02 -1.21818737e-02
   7.38713424e-03 -2.81467776e-09  5.90054210e-09  0.00000000e+00]
 [ 3.03664990e-02  1.59126530e-03 -3.18456366e-03 -1.70196194e-02
  -1.29417706e-02  4.37228857e-10 -4.49005682e-10  0.00000000e+00]
 [ 6.68249145e-02  3.78806734e-02  9.84362337e-03  8.91461502e-03
   7.10909211e-03  4.16964031e-09 -2.25639930e-09  0.00000000e+00]
 [-4.26994603e-02 -6.75800003e-02 -2.67271335e-02 -1.06633286e-02
   5.11453835e-03 -7.18624466e-09  1.49387801e-09  0.00000000e+00]
 [-2.23652208e-02 -3.81322868e-02 -3.90877182e-02 -1.08847636e-02
  -1.00307881e-03 -6.02024210e-09  4.91803601e-10  0.00000000e+00]
 [ 4.89182469e-02 -1.23610170e-03  4.01811186e-02 -6.87178561e-03
  -1.42718922e-02  4.35494631e-09 -1.18060687e-09  0.00000000e+00]
 [-5.70580406e-02 -5.08082980e-02 -1.34877929e-02 -4.10398393e-02
   1.46181929e-02 -5.75261103e-09  5.41129011e-09  0.00000000e+00]
 [ 3.07678740e-02 -4.06150703e-03  6.07937604e-03 -2.85313623e-04
  -1.88851503e-02  1.14177490e-09 -1.99590762e-09  0.00000000e+00]
 [ 7.18403119e-02  4.14268491e-02  2.24812984e-02  1.14451156e-02
```

```
  5.08995235e-02  4.74892597e-09 -1.22691884e-10  0.00000000e+00]
[ 1.53025844e-02  1.92559664e-03 -3.91431043e-02  2.02572059e-02
 -2.37517898e-02 -2.31554736e-09 -4.19363106e-09  0.00000000e+00]
[-5.44491355e-02 -4.02239075e-02 -2.66603066e-02  1.89836195e-03
 -3.41723029e-03 -5.49107219e-09  7.46325196e-10  0.00000000e+00]
[-2.67802405e-02 -8.20431726e-02  6.70012247e-02 -1.50658615e-02
  1.48753217e-03  1.60142481e-10  2.82429052e-09  0.00000000e+00]
[-1.91387908e-02 -5.60290160e-02  2.52562367e-04 -1.24026154e-03
  4.91489107e-04 -3.65592013e-09  2.80604512e-10  0.00000000e+00]
[-5.89174232e-02 -5.76905906e-02  2.15588561e-03  7.79590897e-03
 -4.22926995e-03 -4.10996483e-09  7.98314134e-10  0.00000000e+00]
[ 8.01896312e-02  5.01929005e-02  1.71359070e-02  1.55571621e-02
  1.73928857e-02  5.61279272e-09 -2.49568524e-09  0.00000000e+00]
[ 1.37735218e-02 -8.96111912e-03 -1.48926266e-02 -7.46929049e-03
  8.29444594e-03 -1.76962992e-09  1.31099019e-10  0.00000000e+00]
[-6.27825793e-02 -3.80532110e-02  2.36784340e-03 -2.11923502e-02
  4.10707477e-03 -3.39204619e-09  3.92118624e-09  0.00000000e+00]
[ 9.39865991e-02  5.53060700e-02  7.65367847e-02 -5.91142244e-02
 -1.71920606e-02  1.11591685e-08  2.54410661e-09  0.00000000e+00]
[-4.82657461e-02 -6.32178071e-02 -4.49509812e-02 -2.07031314e-02
  1.00636378e-02 -8.73591583e-09  2.44363639e-09  0.00000000e+00]
[ 1.03721300e-01  7.12600944e-02  2.65194995e-02  2.32891549e-02
  1.16425251e-02  8.23048015e-09 -3.88818747e-09  0.00000000e+00]
[ 3.29674421e-02  1.88827852e-02 -9.83570279e-03  2.36562867e-03
 -2.33351184e-02  1.30540155e-09 -2.61216140e-09  0.00000000e+00]
[ 2.01367338e-02 -2.52298411e-02  2.69229137e-02 -3.22887664e-02
 -3.36791649e-03  9.00791019e-10  1.99825655e-09  0.00000000e+00]
[ 1.21993282e-01  9.77160398e-02  1.85176960e-02  6.30356758e-02
  3.03843437e-02  9.43609478e-09 -6.73404042e-09  0.00000000e+00]
[-6.41521416e-02  3.40775209e-02  1.92783962e-02 -2.58844300e-02
 -7.52424739e-03  2.47198465e-09  4.81884059e-09  0.00000000e+00]
[-1.98794076e-02 -4.23236866e-02 -4.37204147e-03 -2.80239039e-02
  3.86362717e-03 -3.53504130e-09  2.69893213e-09  0.00000000e+00]
[-3.25051031e-02 -4.12532135e-02  6.92423765e-03 -1.93742986e-02
  7.34254807e-04 -2.60296590e-09  2.54195714e-09  0.00000000e+00]
[ 7.15459184e-02  5.76484245e-02 -1.62562924e-02  6.04472348e-02
  1.88570147e-02  3.44905574e-09 -6.34713546e-09  0.00000000e+00]
[ 4.67788600e-02  3.17011434e-02 -2.30389024e-02  3.85488202e-02
  2.80797615e-03  1.01895450e-09 -4.86669001e-09  0.00000000e+00]
[-6.07821841e-02  1.91138873e-02  4.46543431e-02 -4.52189376e-02
  6.10394826e-04  3.46542594e-09  7.04271015e-09  0.00000000e+00]
[ 2.91463041e-02  4.37760554e-03 -1.21678840e-02 -5.60956668e-03
 -1.71761932e-02 -2.26588689e-13 -1.70025921e-09  0.00000000e+00]
[ 1.17327081e-01  8.03565184e-02  3.21379699e-02  1.54899554e-02
 -2.17113737e-02  1.00238093e-08 -5.25574336e-09  0.00000000e+00]
[ 7.27795913e-02  4.20210346e-02  1.41602250e-02  6.95344286e-03
  3.63163262e-03  4.92871329e-09 -2.36640569e-09  0.00000000e+00]
[-8.51762652e-02 -3.29237853e-02  1.93159284e-02 -1.83724643e-03
```

```
  -3.36779651e-03 -1.73753151e-09  3.18842938e-09  0.00000000e+00]
 [-1.71228036e-02 -2.58748646e-02 -5.37973578e-02  1.37697863e-02
  -1.23034779e-02 -6.02406767e-09 -2.45600292e-09  0.00000000e+00]
 [-1.27480373e-01 -7.65600997e-02  2.17470764e-02  2.65861074e-02
  -3.08489528e-03 -4.61138488e-09  2.07263997e-09  0.00000000e+00]
 [-1.67022160e-02 -3.63308458e-02 -3.21296163e-03  3.29699156e-03
  -9.84257953e-03 -2.50693651e-09 -5.79106448e-10  0.00000000e+00]
 [ 8.25169970e-02  6.42112791e-02 -5.44986509e-03  5.38171603e-02
   1.63563729e-02  4.92732094e-09 -6.05849127e-09  0.00000000e+00]
 [-1.53702760e-01 -1.42590090e-02 -2.25700848e-02 -1.17171607e-02
   6.68505654e-03 -5.60117271e-09  6.37868140e-09  0.00000000e+00]
 [ 3.06557249e-02 -2.88451887e-02  5.29364669e-02 -1.39375603e-02
  -1.27076686e-02  3.38827485e-09  1.30743932e-10  0.00000000e+00]
 [-3.82464002e-02 -5.85819478e-02 -3.31044006e-02 -1.91688039e-02
   6.89992923e-03 -7.21870552e-09  2.07222152e-09  0.00000000e+00]
 [ 4.80781732e-02  8.06293700e-03  2.44527694e-02 -4.92533697e-04
   2.30372522e-03  3.31677907e-09 -1.04657529e-09  0.00000000e+00]
 [-1.15453385e-01 -2.24727267e-02 -4.65931691e-02  4.89088771e-04
   1.26696041e-03 -7.25780609e-09  3.13011677e-09  0.00000000e+00]
 [ 1.53081814e-02 -1.36515501e-03 -3.13712769e-02  1.04643300e-02
  -1.70840606e-02 -2.06189341e-09 -2.93412967e-09  0.00000000e+00]
 [ 2.72641734e-02  5.90187563e-02  1.44127911e-03  1.79763265e-02
  -3.66077632e-02  4.92224448e-09 -3.71948726e-09  0.00000000e+00]
 [ 7.96247782e-03 -1.93017922e-02 -5.95170635e-03  3.93509136e-03
   2.20173832e-02 -1.88416438e-09  1.97163246e-10  0.00000000e+00]
 [ 1.26809060e-02  1.99997346e-02  3.34459702e-02 -4.45631819e-02
  -6.09178032e-03  3.95710585e-09  3.71398300e-09  0.00000000e+00]
 [ 4.10122593e-02  3.38587102e-02 -5.06055170e-03  8.41363638e-03
  -2.93894140e-02  2.89882925e-09 -3.46438053e-09  0.00000000e+00]
 [ 3.68393007e-02  4.97856572e-02  5.00745904e-02 -3.94601249e-02
  -1.66800987e-02  7.78122574e-09  2.49794957e-09  0.00000000e+00]
 [-4.85257764e-02 -3.98998709e-02  4.89441853e-02 -1.46836930e-02
  -2.63860603e-03  8.49737230e-10  3.44218730e-09  0.00000000e+00]
 [ 1.19509485e-02  1.76957122e-02  1.38734574e-02 -2.54048762e-02
  -1.18188800e-02  2.42607105e-09  1.47476271e-09  0.00000000e+00]
 [-6.45970171e-03 -3.91826887e-02 -3.48021326e-03 -1.97960445e-02
  -1.25471126e-03 -2.87931409e-09  1.32248749e-09  0.00000000e+00]
 [ 4.07862763e-02  2.63447101e-02 -1.81317743e-02  3.98128509e-02
   4.17476446e-02  3.07226578e-10 -2.71519543e-09  0.00000000e+00]
 [ 5.21428942e-03  2.98265738e-02  5.24283287e-03 -8.28205684e-03
  -1.94035343e-02  2.58997529e-09 -1.07362809e-10  0.00000000e+00]
 [ 1.91027602e-02  7.56070797e-03 -3.53828360e-02  3.14338157e-02
   9.27774746e-03 -2.10986355e-09 -3.42467504e-09  0.00000000e+00]
 [-1.93128625e-02 -3.94396877e-02 -3.10176763e-02 -5.95389362e-03
  -3.54916231e-03 -5.27619067e-09 -1.76181658e-12  0.00000000e+00]
 [ 4.23402619e-02  1.77939116e-02 -1.77992227e-03  7.39845121e-03
   2.20011270e-02  1.29020596e-09 -9.02527412e-10  0.00000000e+00]
 [ 5.69786878e-03  2.46298837e-02  2.80547848e-02 -3.61546697e-02
```

```
  -9.15672917e-03  3.78654885e-09  3.07011304e-09  0.00000000e+00]
 [-4.25461645e-02  3.32482766e-02  1.48194859e-02 -1.99628143e-02
  -1.14761573e-02  2.53917931e-09  3.23911553e-09  0.00000000e+00]
 [ 6.71463125e-02  4.84273535e-02 -4.92904486e-03  4.35790361e-02
   3.38442004e-02  3.36136019e-09 -3.91284656e-09  0.00000000e+00]
 [ 1.70635892e-01  1.45049114e-01  3.84986386e-02  9.44342869e-02
   5.95513360e-02  1.44988207e-08 -8.72042959e-09  0.00000000e+00]
 [ 4.50192379e-02  9.01753301e-03  2.01697210e-02 -2.73265702e-02
   1.59678806e-02  2.47934660e-09  1.86664649e-09  0.00000000e+00]
 [-7.49881347e-02 -2.77852540e-03 -3.21904391e-02  2.22737176e-03
  -7.67203643e-03 -4.00168271e-09  1.52324737e-09  0.00000000e+00]
 [ 9.24958475e-03 -1.28009910e-02 -2.08963748e-02 -9.95466529e-03
  -1.00054949e-02 -2.26651944e-09 -5.92864790e-10  0.00000000e+00]
 [ 2.57825826e-03  1.23554176e-02 -8.30779662e-03 -6.29681526e-03
  -1.59964039e-02  3.17913697e-10 -4.44604152e-10  0.00000000e+00]
 [-1.76950590e-02 -5.00260525e-02 -8.92734217e-03 -3.61628832e-04
  -5.56016033e-03 -3.92992261e-09 -2.78323649e-10  0.00000000e+00]
 [ 5.18188159e-02  3.42819869e-02 -2.20006258e-02  3.20433826e-02
  -2.28231534e-02  1.76143954e-09 -5.78582525e-09  0.00000000e+00]
 [ 1.47641917e-01  1.03700499e-01  6.28770427e-02  1.84225660e-02
   2.82746970e-02  1.36285644e-08 -3.21172007e-09  0.00000000e+00]
 [-1.14754474e-02 -2.06590419e-02 -5.20678790e-02  1.60368677e-02
  -1.47525972e-02 -5.40724373e-09 -2.88711291e-09  0.00000000e+00]
 [-8.74373934e-02 -3.88696394e-02 -1.79049064e-02 -1.31014819e-02
   4.80760997e-03 -5.51541337e-09  3.79425680e-09  0.00000000e+00]
 [-8.43520914e-02 -1.12325061e-02  5.27488888e-02 -3.73658212e-02
   4.48758734e-03  1.93736508e-09  7.33517171e-09  0.00000000e+00]
 [-9.86773822e-02  5.21371118e-02 -2.31973902e-02  2.91450107e-02
  -2.60543421e-02  1.84294893e-10  2.83934401e-12  0.00000000e+00]
 [ 3.87915833e-02  2.56538500e-02 -3.10955513e-02  3.82085497e-02
  -9.10598930e-03  5.95139090e-11 -5.37438238e-09  0.00000000e+00]
 [-2.52022230e-02 -3.85898235e-02  2.94034519e-02 -5.06178056e-02
   1.03137034e-02 -8.67260880e-10  5.77862680e-09  0.00000000e+00]
 [ 1.18171269e-01  8.21451295e-02  3.61571225e-02  2.15186447e-02
   7.33846993e-03  1.00012595e-08 -4.19547529e-09  0.00000000e+00]
 [-9.60049189e-02 -6.65754126e-02  7.12729436e-02  7.57911613e-03
  -3.71554306e-03  5.46070529e-10  3.51901239e-09  0.00000000e+00]
 [ 3.26234619e-02  4.42918981e-03 -4.15094668e-03 -1.38225459e-02
  -1.48376770e-02  6.25180993e-10 -8.80130639e-10  0.00000000e+00]
 [-8.85715649e-02 -5.60186443e-02 -1.37873824e-02  6.96896963e-04
   2.18806434e-03 -6.04071619e-09  2.49393873e-09  0.00000000e+00]
 [-4.30394109e-02  6.81177777e-03 -2.41639637e-02  2.99124271e-03
  -1.32356386e-02 -2.09985003e-09  2.53430122e-10  0.00000000e+00]
 [ 4.31402094e-02  1.86372899e-02 -2.37720870e-03 -4.69593187e-03
  -2.12142475e-02  1.99433537e-09 -2.15845476e-09  0.00000000e+00]
 [ 2.99916912e-02  2.64750674e-02 -4.71129705e-02  6.46785602e-02
   1.64352199e-02 -1.63590261e-09 -6.21081404e-09  0.00000000e+00]
 [-1.81911363e-02 -4.82316704e-02 -1.22145566e-03  1.25271805e-04
 -6.53367330e-03 -3.15976989e-09 -1.75912643e-10  0.00000000e+00]
[-5.54013259e-02 -1.69254896e-02 -5.48703236e-02  1.80599888e-02
 -1.17878175e-02 -6.20958779e-09 -1.29540343e-09  0.00000000e+00]
[ 3.18842140e-02 -2.75758350e-03  9.93217692e-03 -3.38973059e-02
 -6.59984725e-03  1.04999888e-09  1.41073921e-09  0.00000000e+00]
[ 3.75303551e-02  1.34747848e-02 -9.45136359e-03  3.24386986e-03
 -6.43490951e-03  7.91278521e-10 -2.03594817e-09  0.00000000e+00]
[ 8.24269926e-02  4.48860214e-02  2.83639118e-02 -8.45577571e-03
 -3.70290037e-03  6.46479992e-09 -1.53063930e-09  0.00000000e+00]
[ 3.53225901e-02  1.41764742e-02 -1.39375163e-02  1.16651179e-02
  2.56138256e-03  3.23611016e-10 -2.26283286e-09  0.00000000e+00]
[-5.97201854e-02  9.54942880e-04 -7.37841944e-03 -2.03251222e-02
 -2.60874441e-03 -1.71214788e-09  3.58669817e-09  0.00000000e+00]
[-6.26776875e-02 -6.71610124e-02  4.52638412e-02 -6.25118734e-03
  3.69321088e-04 -1.30612895e-09  3.08823386e-09  0.00000000e+00]
[ 3.82422421e-02  1.54076686e-02 -8.23125398e-03  9.07741796e-03
  1.12169288e-02  7.46762093e-10 -1.59156117e-09  0.00000000e+00]
[-3.53612288e-02 -4.24313080e-02 -3.75764730e-02 -1.65567551e-02
  3.10978953e-03 -6.49454594e-09  1.63059797e-09  0.00000000e+00]
[ 3.77061359e-02  9.47532682e-03  1.41918991e-02 -3.01390012e-03
  7.25110814e-02  1.04630076e-09  2.93129816e-09  0.00000000e+00]
[ 4.70022835e-02  1.34979724e-02  1.11033076e-02 -2.01613716e-02
 -7.82289124e-03  2.51340209e-09 -1.40932381e-10  0.00000000e+00]
[-1.17543967e-01 -2.33477174e-02  4.36125169e-02 -5.63816888e-02
  1.78421910e-02 -5.28724473e-10  1.04836073e-08  0.00000000e+00]
[-1.28453937e-01 -6.72673272e-03  1.48225321e-03 -1.67691225e-02
  3.31991400e-03 -2.67471545e-09  6.24393917e-09  0.00000000e+00]
[-6.32937018e-02 -3.87058503e-02 -4.25311243e-02 -1.09487172e-02
  3.28032622e-03 -7.12352587e-09  2.15368917e-09  0.00000000e+00]
[ 1.03038393e-01  7.12001077e-02  2.28956330e-02  2.38039871e-02
  1.80561988e-04  8.12464549e-09 -4.56331709e-09  0.00000000e+00]
[-3.41993852e-03 -1.41888550e-02 -4.74672819e-02  1.63216691e-02
 -1.70655688e-02 -4.45201023e-09 -3.16889000e-09  0.00000000e+00]
[ 4.05619619e-02  1.94965843e-02 -1.62489156e-02  1.39786616e-02
 -1.49384936e-02  8.75837670e-10 -3.53109000e-09  0.00000000e+00]
[ 1.16254277e-02 -6.32204097e-04 -3.49667640e-02  2.44636591e-02
  1.34442921e-02 -2.82576637e-09 -2.44260328e-09  0.00000000e+00]
[-4.47148644e-02 -6.54790750e-02 -3.62999787e-02  2.56783892e-03
  4.00163416e-04 -7.70335514e-09  8.08888931e-11  0.00000000e+00]
[ 5.03955440e-02  2.77052696e-02 -6.26320311e-03  1.74850341e-02
  7.74949224e-03  1.98709991e-09 -2.73865191e-09  0.00000000e+00]
[ 2.68676501e-02 -1.01730652e-02  8.58453954e-03  5.24302984e-03
 -2.01018562e-02  9.92771022e-10 -2.37319313e-09  0.00000000e+00]
[ 8.31577845e-02  5.97501718e-02  3.73955985e-02 -1.06735485e-02
 -1.70145465e-02  8.34822989e-09 -1.72656894e-09  0.00000000e+00]
[-9.92057898e-03 -1.45857645e-02 -6.17940869e-02  3.07623444e-02
 -2.10717243e-02 -5.59787072e-09 -4.59891459e-09  0.00000000e+00]
[ 1.01554608e-02  3.07749573e-02  1.32954492e-02 -1.57866714e-02
```

```
  -1.76407381e-02  3.31336097e-09  5.81367035e-10  0.00000000e+00]
 [ 2.31957582e-02 -6.75332015e-03 -1.59244166e-03 -2.51340287e-02
  -7.73609128e-03 -2.12146955e-10  6.89913243e-10  0.00000000e+00]
 [-1.34697935e-01  2.42026745e-02 -9.80564310e-03 -7.57532532e-03
  -4.26984919e-03 -1.67681892e-09  5.42102687e-09  0.00000000e+00]
 [-2.18798190e-02 -4.68615653e-02 -1.91931078e-02 -3.48701850e-02
   8.52576466e-03 -5.23322229e-09  3.22279367e-09  0.00000000e+00]
 [ 1.27901580e-01  7.38823236e-02  7.53684363e-02 -2.77832705e-02
   1.53051550e-02  1.24153121e-08  5.64735170e-10  0.00000000e+00]]
Correlation Coefficients:
0.9718990329301156
```

```
[ ]:
```