w4oo3pozb

November 25, 2024

1 Table of contents

Imports

[1]:
```
# Data Processing
import numpy as np
import pandas as pd

# ML models
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler, LabelEncoder, MinMaxScaler

# Dimension Reduction
from sklearn.decomposition import PCA
from sklearn.manifold import MDS, TSNE, LocallyLinearEmbedding

# Statistics
from scipy.spatial import procrustes
from scipy.stats import pearsonr
from sklearn.metrics import accuracy_score

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Image processing
from PIL import Image, ImageOps
```

```
import os
import cv2

# misc
import time
import warnings
warnings.filterwarnings("ignore")
```

Nice Visualizations

[2]:
```
plt.rc('font', family='Serif', size=12)
plt.rc('axes', titlesize=14, titleweight='bold')
plt.rc('axes', labelsize=12, titlesize=14)
plt.rc('legend', fontsize=10)
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)
```

Global Variables and file paths

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

RANDOM_STATE = 99
COLOR_MAP = plt.cm.twilight
IMAGE_FOLDER = "/kaggle/input/assignment-data/Data/Rock_Images"
VALIDATION_FOLDER = "/kaggle/input/assignment-data/Data/Validation_Images"
HUMAN_PREDICTIVE_FEATURES = "/kaggle/input/assignment-data/Data/mds_360.txt"
HUMAN_PREDICTIVE_FEATURES_VALIDATION = "/kaggle/input/assignment-data/Data/
 ↪mds_120.txt"
```

Pre Processing Load the rock images

[4]:
```
def get_label_from_filename(filename):
    if filename.startswith('I'):
        return 'Igneous'
    elif filename.startswith('M'):
        return 'Metamorphic'
    else:
        return 'Sedimentary'

def load_images_to_dataframe(folder):
    data = []  # To store flattened image data
    filenames = []  # To store filenames
    labels = []  # To store labels

    for filename in os.listdir(folder):
        if filename.endswith(".jpg"):  # Process only .jpg files
            # Assign label
```

```
            label = get_label_from_filename(filename)
            labels.append(label)

            # Load and preprocess image
            img_path = os.path.join(folder, filename)
            img = Image.open(img_path)
            img = ImageOps.grayscale(img)
            img = img.resize((256, 256))
            img_array = np.array(img) / 255.0

            # Flatten and append data
            data.append(img_array.flatten())
            filenames.append(filename)

    # Create DataFrame with image data
    image_df = pd.DataFrame(data, columns=list(range(len(data[0]))))

    # image_df['filename'] = filenames
    # image_df['label'] = labels

    return image_df, pd.Series(labels)
```

[5]:
```
DATA, LABELS = load_images_to_dataframe(IMAGE_FOLDER)

rock_mapping = {"Igneous": 0, "Metamorphic": 1, "Sedimentary": 2}
encoded_labels = LABELS.map(rock_mapping)
```

Total Number of Rocks

[6]:
```
len(DATA)
```

[6]: 360

[7]:
```
print(encoded_labels)
```

```
0      0
1      0
2      2
3      2
4      0
      ..
355    2
356    2
357    1
358    0
359    1
Length: 360, dtype: int64
```

The total number of features after converting to grayscale, resizing and flatenning image is 65536.

[8]:
```
len(DATA.columns)
```

[8]:
```
65536
```

[9]:
```
print(DATA.shape)
```

```
(360, 65536)
```

[10]:
```
DATA.head()
```

[10]:

	0	1	2	3	4	5	6	7	8	9	…	\
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	…	
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	…	
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	…	
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	…	
4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	…	

	65526	65527	65528	65529	65530	65531	65532	65533	65534	65535
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

[5 rows x 65536 columns]

Pre Processing Details The following preprocessing steps have been done over the images: 1. Converted to Grayscale 2. Resized to (256, 256) 3. Dividing the pixel values by 255. 4. Mapping the rock labels to integer values.

Queston 1

[11]:
```
pca = PCA(n_components = 0.9, svd_solver = "full")
X_reduced = pca.fit_transform(DATA)
X_recovered = pca.inverse_transform(X_reduced)
```

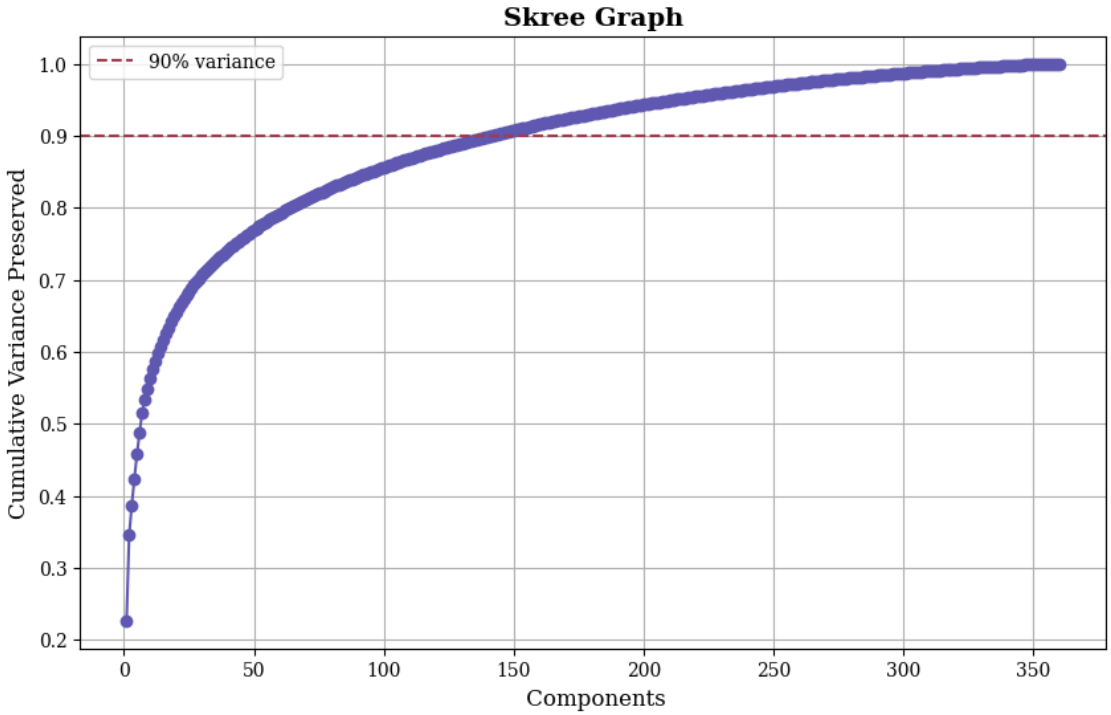
[12]:
```
print("Length of reduced vector to preserve 90% variance: ", len(X_reduced[0]))
```

```
Length of reduced vector to preserve 90% variance:  141
```

[13]:
```
# Getting cumulative variance preserved for every feature included in the PCA.
line_color = COLOR_MAP(.3)
hline_color = COLOR_MAP(.7)

pca = PCA(svd_solver = "full")
pca.fit(DATA)
cumulative_variance = pca.explained_variance_ratio_.cumsum()
```

```
n_components_90 = (cumulative_variance >= 0.90).argmax() + 1

print(f"Number of components required to preserve 90% variance:␣
 ↪{n_components_90}")
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,␣
 ↪marker='o', color=line_color)
plt.xlabel(' Components')
plt.ylabel('Cumulative Variance Preserved')
plt.title('Skree Graph')
plt.axhline(y=0.90, color='r', linestyle='--', label="90% variance",␣
 ↪c=hline_color)
plt.legend()
plt.grid(True)
plt.show()
```

```
Number of components required to preserve 90% variance: 141
```



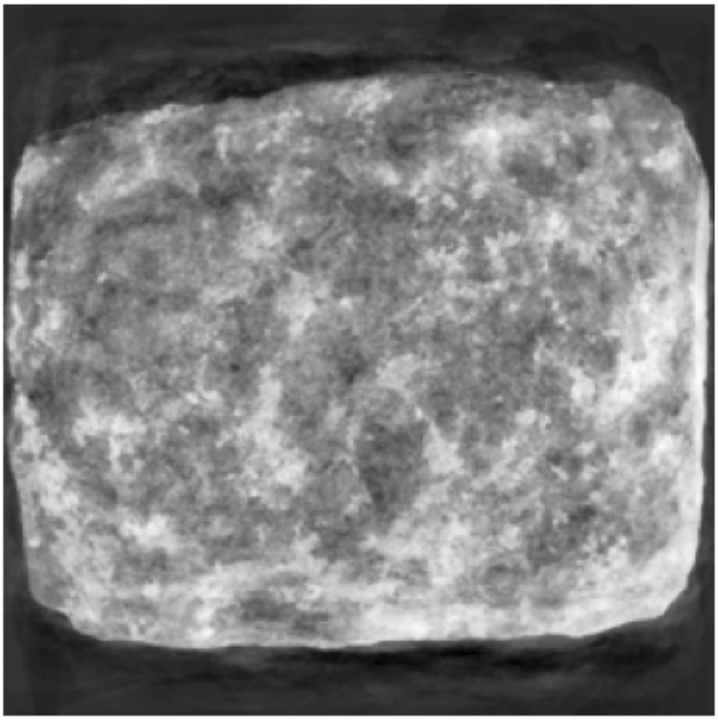
Answer 1 1. PCA has been applied to the data. 2. 141 components are required to capture 90% of the variance in the data.

Question 2

[14]:
```
plt.imshow(np.array(DATA.iloc[0, : ]).reshape((256, 256)), cmap = 'Greys')
plt.axis('off')
plt.show()
```

[15]:
```
pca = PCA(n_components = 0.9, svd_solver = "full")
X_reduced = pca.fit_transform(DATA)
X_recovered = pca.inverse_transform(X_reduced)
```

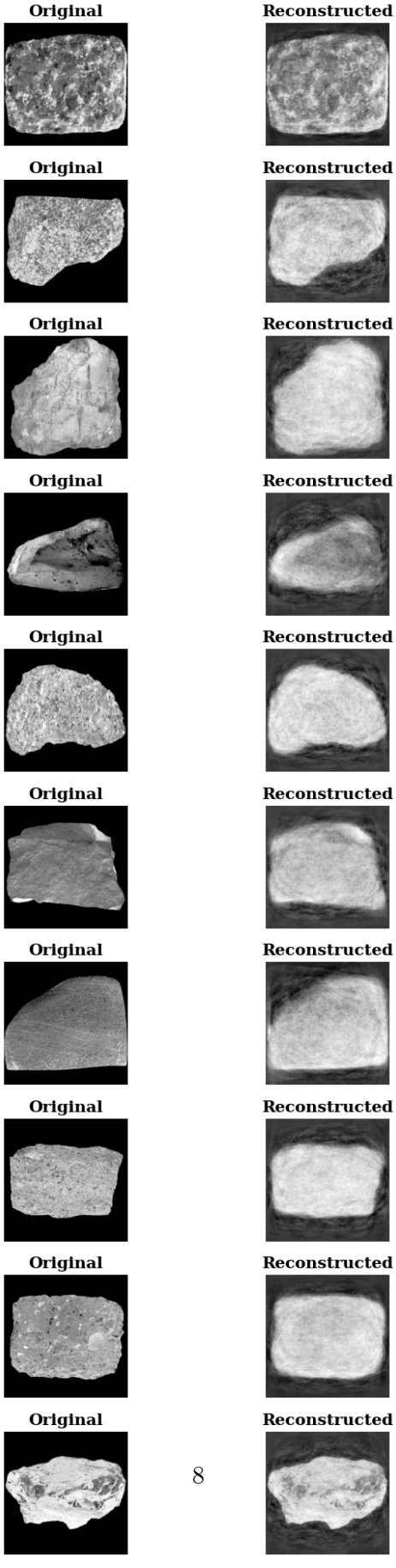
[16]:
```
plt.imshow(X_recovered[0].reshape((256, 256)), cmap = "Greys")
plt.axis('off')
plt.show()
```



[17]:
```
# Plotting the images pre and post reconstruction side by side:

num_images = 10
sample_indices = range(num_images)

plt.figure(figsize = (10, 15))

fig, axes = plt.subplots(num_images, 2, figsize=(8, num_images * 2))

for i, idx in enumerate(sample_indices):
    axes[i, 0].imshow(np.array(DATA.iloc[idx, : ]).reshape((256, 256)),␣
 ↪cmap='Greys')
    axes[i, 0].set_title("Original")
    axes[i, 0].axis('off')

    axes[i, 1].imshow(X_recovered[idx].reshape((256, 256)), cmap='Greys')
    axes[i, 1].set_title("Reconstructed")
    axes[i, 1].axis('off')

plt.suptitle("Original vs. Reconstructed Images")
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.show()
```

```
<Figure size 1000x1500 with 0 Axes>
```

Original vs. Reconstructed Images



Answer 1. Original and Reconstructed images have been plotted.

Question 3

3.A

[18]:
```
pca_two = PCA(n_components = 2)
X_reduced_two = pca_two.fit_transform(DATA)
X_recovered_two = pca_two.inverse_transform(X_reduced_two)
```

[19]:
```
pca_two.explained_variance_ratio_
```

[19]:
```
array([0.22656558, 0.11828768])
```

[20]:
```
print("Total variance explained by the first two components: {0}%".
  ↪format(sum(pca_two.explained_variance_ratio_) * 100))
```

```
Total variance explained by the first two components: 34.485326719873214%
```

[21]:
```
plot_data = pd.DataFrame(columns = ["PC1", "PC2", "Label"])

for itr in range(len(X_reduced_two)):
  curr_row = [X_reduced_two[itr][0], X_reduced_two[itr][1], LABELS[itr]]
  plot_data.loc[len(plot_data)] = curr_row
```

[22]:
```
plot_data
```

[22]:

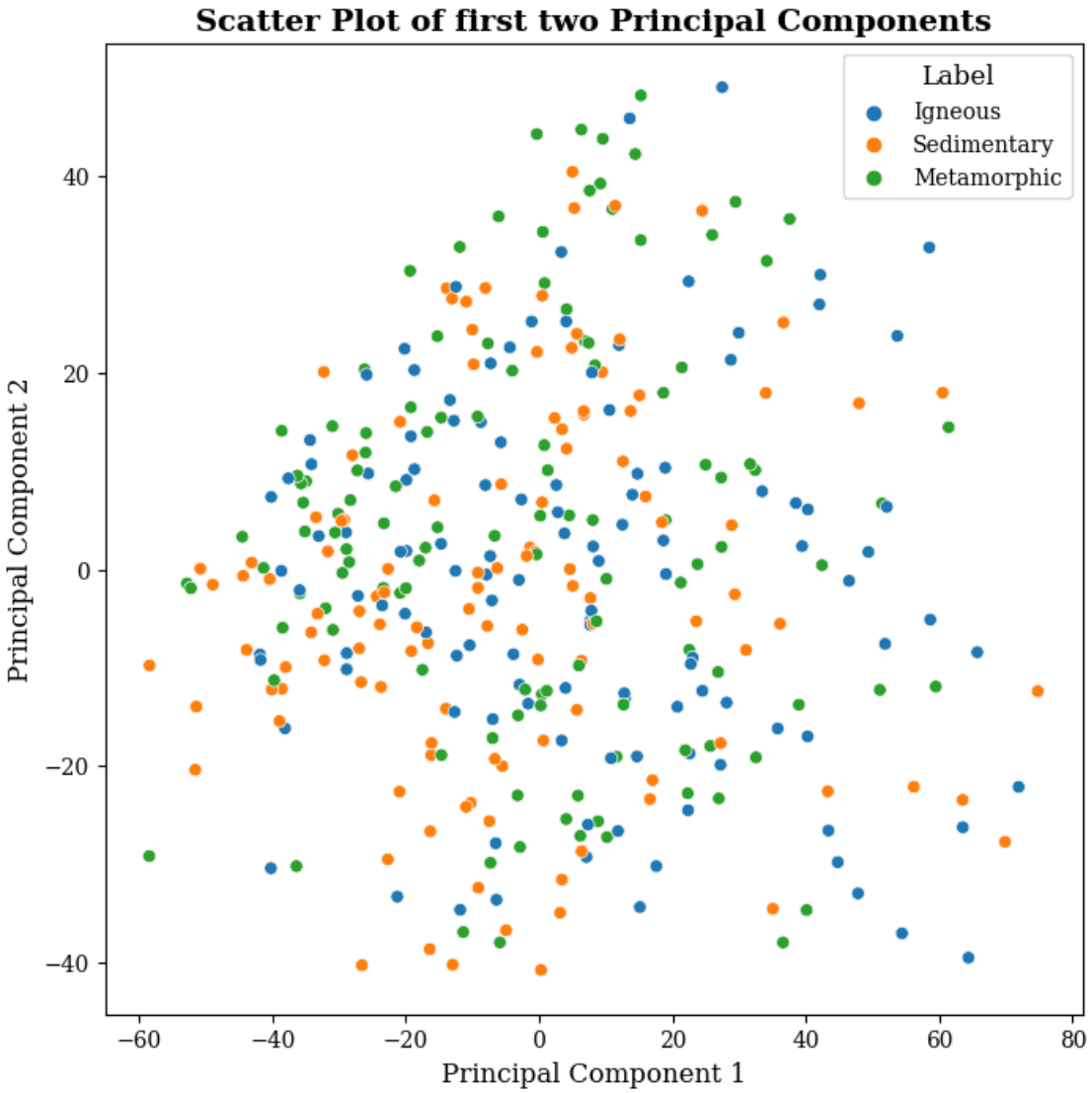
	PC1	PC2	Label
0	7.123492	-29.280801	Igneous
1	-3.001944	-1.072161	Igneous
2	17.026849	-21.458015	Sedimentary
3	-51.374545	-13.971027	Sedimentary
4	-2.630438	7.119230	Igneous
..	…	…	…
355	11.428866	37.024257	Sedimentary
356	-10.897358	27.254740	Sedimentary
357	-58.426701	-29.189459	Metamorphic
358	35.782668	-16.196399	Igneous
359	8.638412	-5.279535	Metamorphic

[360 rows x 3 columns]

3.B

[23]:
```
plt.figure(figsize = (8, 8))

sns.scatterplot(data = plot_data, x = 'PC1', y = 'PC2', hue = 'Label')
```

```
plt.title('Scatter Plot of first two Principal Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Label')
plt.show()
```



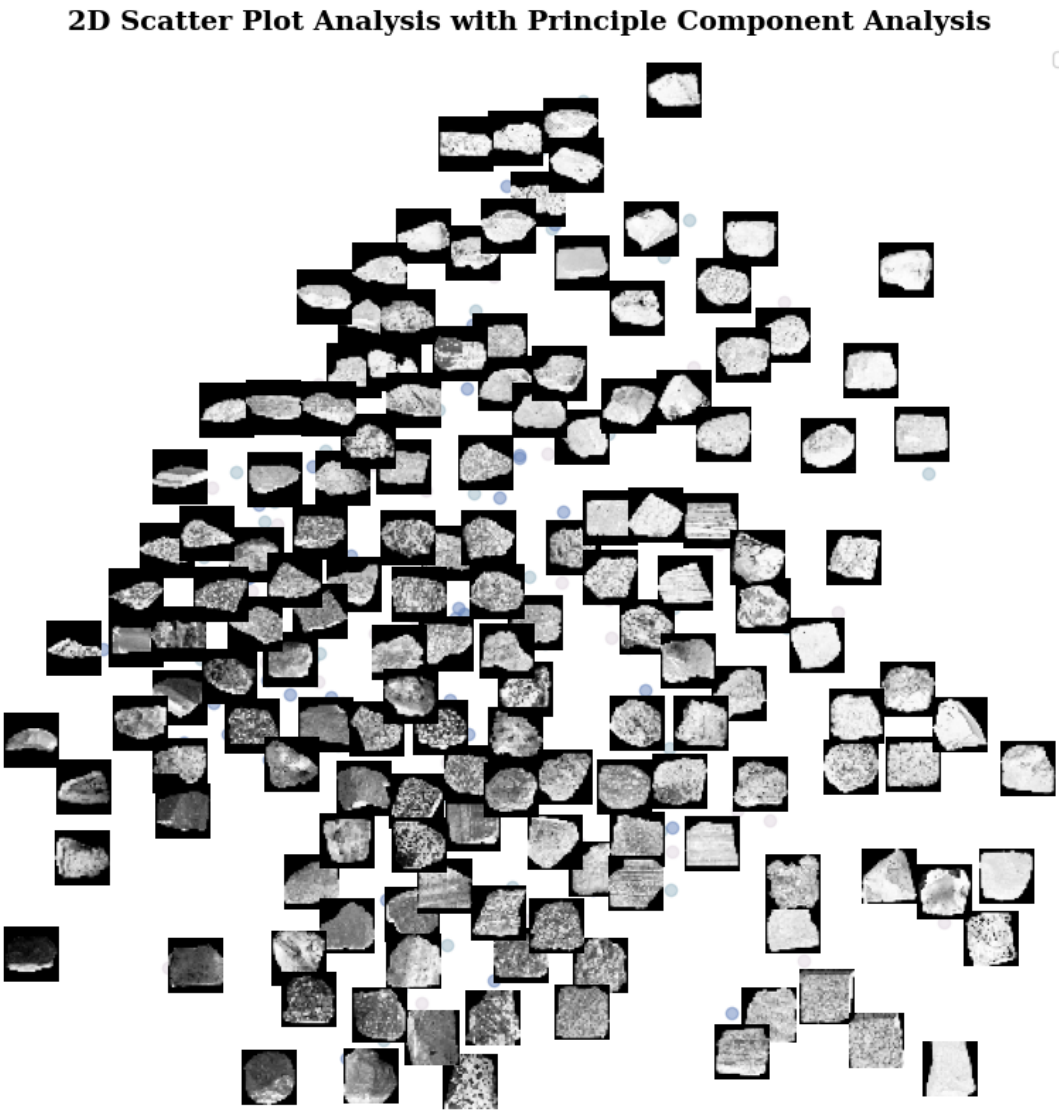
[24]:
```
from sklearn.preprocessing import MinMaxScaler
from matplotlib.offsetbox import AnnotationBbox, OffsetImage

def plot_digits(X, y, min_distance=0.04, images=None, figsize=(13, 10),␣
 ↪method=None):
```

```
    X_normalized = MinMaxScaler().fit_transform(X)

    neighbors = np.array([[10., 10.]])

    plt.figure(figsize=figsize)
    plt.title(method)
    plt.legend()

    digits = np.unique(y)
    for digit in digits:
        plt.scatter(X_normalized[y == digit, 0], X_normalized[y == digit, 1],
                    c=[COLOR_MAP(float(digit) / 9)], alpha=0.5)
    plt.axis("off")
    ax = plt.gca()
    for index, image_coord in enumerate(X_normalized):
        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 None:
                plt.text(image_coord[0], image_coord[1], str(int(y[index])),
                         color=COLOR_MAP(float(y[index]) / 9),
                         fontdict={"weight": "bold", "size": 16})
            else:
                image = np.array(images.iloc[index, : ]).reshape((256, 256))
                image = cv2.resize(image, (28, 28))
                imagebox = AnnotationBbox(OffsetImage(image, cmap='Greys'),
                                          image_coord, frameon=False,␣
↪arrowprops=dict(arrowstyle="->"))
                ax.add_artist(imagebox)


def plot_principle_components(Reduced_X, method):
    print("Total variance explained by the first two components: ", sum(pca_two.
↪explained_variance_ratio_))

    plot_data = pd.DataFrame(
        {
            "PC1": Reduced_X[:, 0],
            "PC2": Reduced_X[:, 1],
            "Label": LABELS
        }
    )

    plot_digits(
        plot_data[["PC1", "PC2"]],
        encoded_labels,
        images=DATA,
```

```
        figsize=(10, 10),
        method = method
    )
```

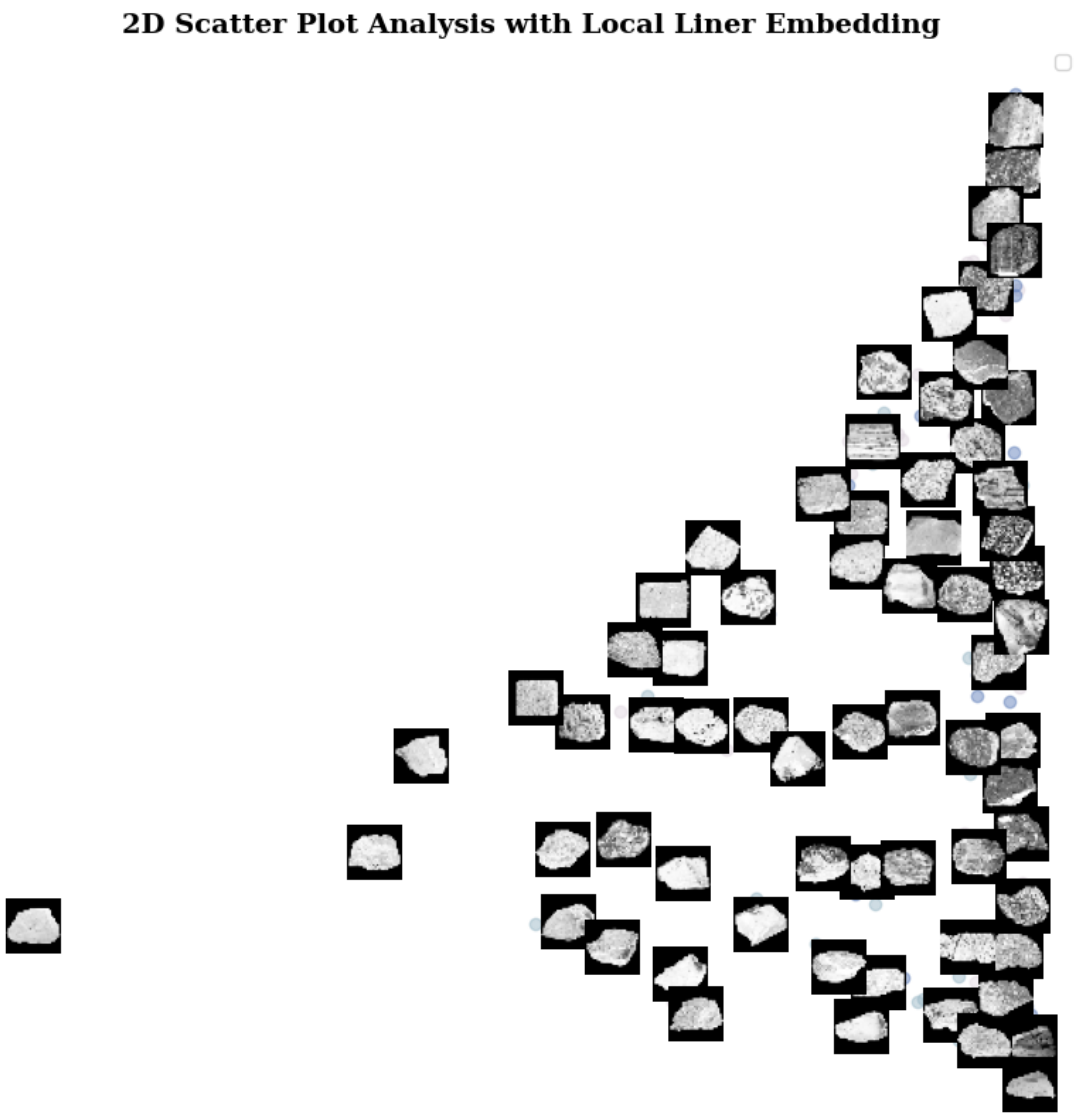
[25]:
```
pca_two = PCA(n_components = 2)
Reduced_X = pca_two.fit_transform(DATA)
method = '2D Scatter Plot Analysis with Principle Component Analysis'
plot_principle_components(Reduced_X, method)
```

```
Total variance explained by the first two components:  0.3448532671987312
```



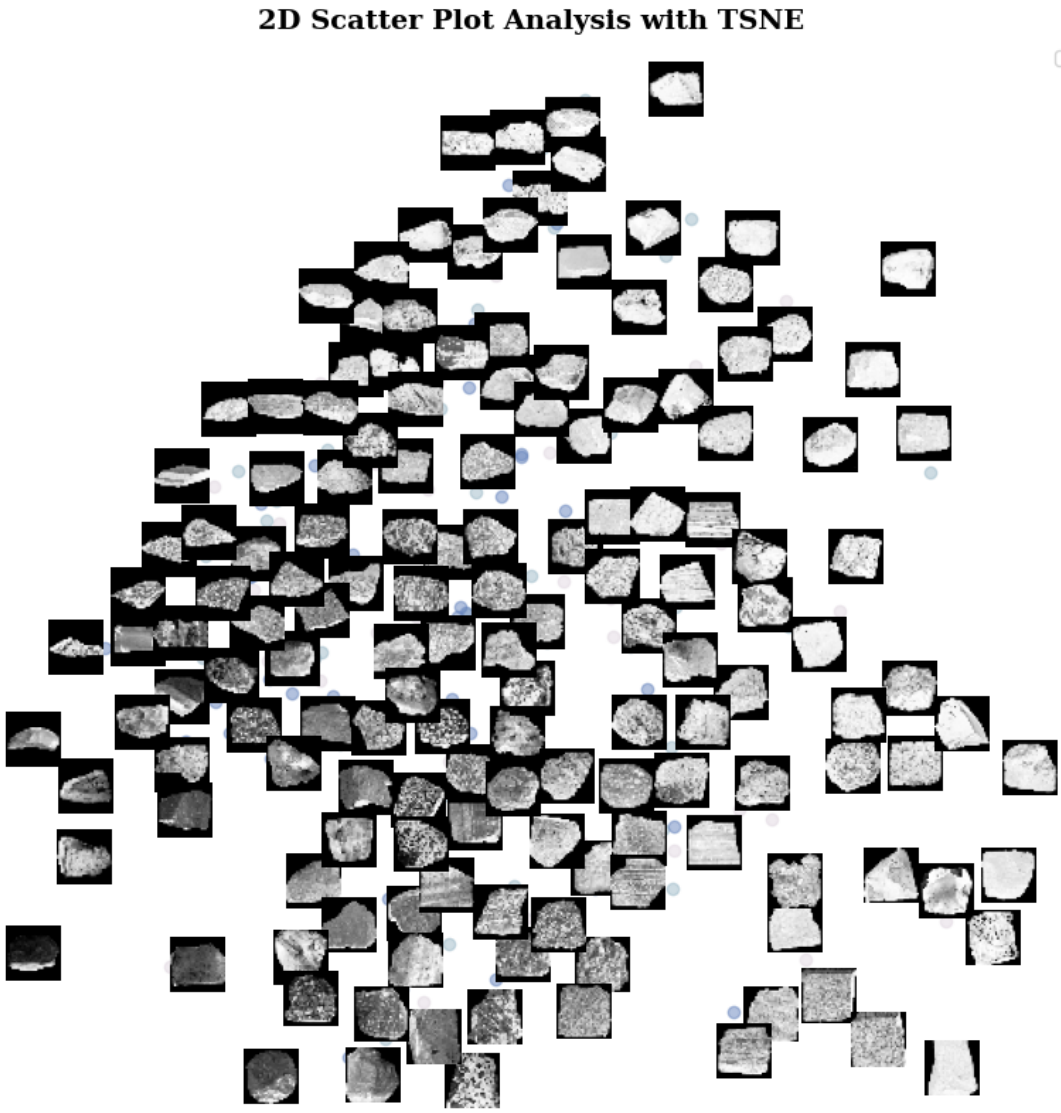
[26]:
```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=5)
Reduced_X = lle.fit_transform(DATA)
method = '2D Scatter Plot Analysis with Local Liner Embedding'
plot_principle_components(Reduced_X, method)
```

```
Total variance explained by the first two components:  0.3448532671987312
```



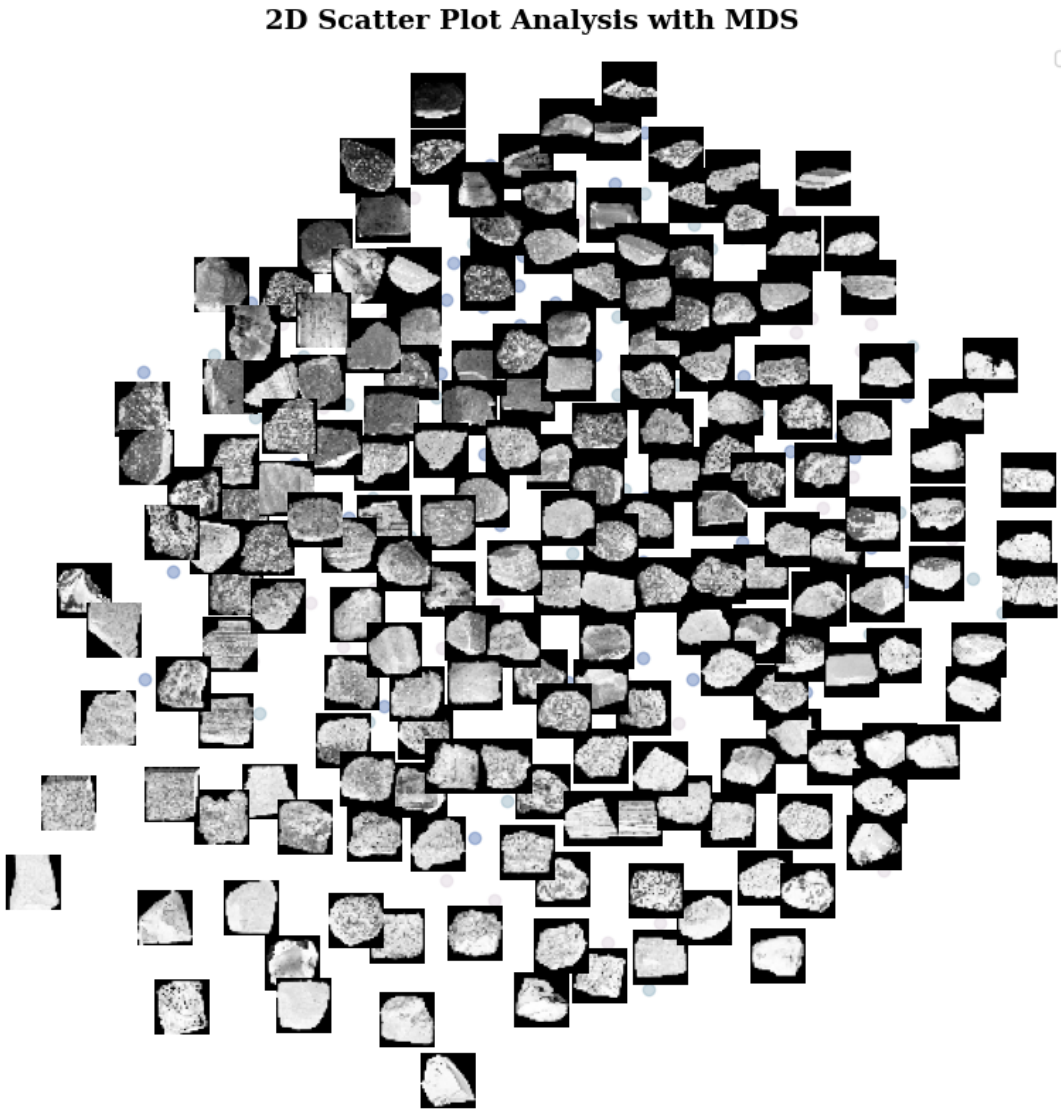
[27]:
```
tsne = TSNE(n_components=2, perplexity=1, random_state=0)
Reduced_X = pca_two.fit_transform(DATA)
method = '2D Scatter Plot Analysis with TSNE'
plot_principle_components(Reduced_X, method)
```

```
Total variance explained by the first two components:  0.344853267198732
```



[28]:
```
mds = MDS(n_components=2, random_state=0)
Reduced_X = mds.fit_transform(DATA)
method = '2D Scatter Plot Analysis with MDS'
plot_principle_components(Reduced_X, method)
```

```
Total variance explained by the first two components:  0.344853267198732
```



Answer 3A: 1. Total variance explained by the first two principal components: 34.48532671987328%

3B: 1. 5 2D scatter plots have been built:

`a. First 2 components using PCA.`

`b. Scatter Plots with first two components with images for PCA, LLE, TSNE and MDS`

3C: Discussion over plots:

1. In all the plots, there does not seem to be a clear distinguishing boundary between the 3 types of rock categories.
2. One of the main reasons for this can be the information loss when we reduce the images to

only 2 dimensions.

3. For PCA and TSNE, the darker shade rocks seem to be on the left side of the chart while the lighter shades are on the right side. For MDS, this is completely opposite with the lighter ones found on the left and the darker ones on the right.
4. For LLE, there does not seem to be a great degree of separation between the shades of the images.
5. For MDS, there is more compactness between the similar looking images i.e. the dark and light shade rocks are tightly packed closer than seen in other transformations.
6. The plots show that there is no distinct boundary between the three different types of rocks based on the first two components. This can mean that the rock categories gradually transition from one type to other, which is actually the case as igneous rocks transform into metamorphic and then sedimentary rocks.

Question 4

[29]:
```
matrix_with_human_data = np.loadtxt(HUMAN_PREDICTIVE_FEATURES)
print(matrix_with_human_data)
print("Shape of Most important features for humans", matrix_with_human_data.
 ↪shape)


procrustes_analysis = []
```

```
[[-3.743 -1.204  2.001 … -1.992  4.95   1.695]
 [ 2.332  1.625  0.985 …  0.093  6.724  0.708]
 [ 0.346  1.49  -3.795 … -3.786  0.706 -2.854]
 …
 [-3.475 -3.431 -2.184 … -2.265  1.129 -1.201]
 [-0.051 -2.358  1.994 …  7.268 -0.593 -1.432]
 [ 1.134 -4.9    0.983 …  4.695  0.624 -1.195]]
Shape of Most important features for humans (360, 8)
```

PCA

[30]:
```
mds = PCA(n_components=8, random_state=RANDOM_STATE)
matrix_with_pca_embeddings_data = mds.fit_transform(DATA)
mtx1, mtx2, disparity = procrustes(matrix_with_human_data,␣
 ↪matrix_with_pca_embeddings_data)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

pca_result = {
        'Embedding': 'PCA',
        'Disparity': disparity,
        'Correlation': stat,
        'P_Value': p_value
```

```
}

procrustes_analysis.append(pca_result)

print('Embedding: PCA')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: PCA
Disparity: 0.9903066734529975
Correlation: 0.09845469286429505
P_value: 1.1953461896430938e-07
```

[31]:
```
pca_corr_table = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  pca_corr_table.loc[len(pca_corr_table)] = curr_row

pca_corr_table = pca_corr_table.sort_values(by = ['Correlation'], ascending =␣
 ↪False).reset_index(drop = True)

print(pca_corr_table)
```

```
     Dimension  Correlation
0  Dimension 1     0.140161
1  Dimension 7     0.116476
2  Dimension 8     0.109885
3  Dimension 2     0.104116
4  Dimension 3     0.095783
5  Dimension 4     0.085307
6  Dimension 5     0.071380
7  Dimension 6     0.070697
```

LLE

[32]:
```
lle = LocallyLinearEmbedding(n_components=8, n_neighbors=10)
matrix_with_lle_embeddings_data = lle.fit_transform(DATA)
mtx1, mtx2, disparity = procrustes(matrix_with_human_data,␣
 ↪matrix_with_lle_embeddings_data)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

lle_result = {
        'Embedding': 'LLE',
        'Disparity': disparity,
        'Correlation': stat,
```

```
        'P_Value': p_value
}

procrustes_analysis.append(lle_result)

print('Embedding: LLE')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: LLE
Disparity: 0.986906780024853
Correlation: 0.11442560891315894
P_value: 7.353328056719625e-10
```

[33]:
```
lle_corr_table = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  lle_corr_table.loc[len(lle_corr_table)] = curr_row

lle_corr_table = lle_corr_table.sort_values(by = ['Correlation'], ascending =␣
 ↪False).reset_index(drop = True)

print(lle_corr_table)
```

```
     Dimension  Correlation
0  Dimension 7     0.160246
1  Dimension 2     0.153362
2  Dimension 8     0.146734
3  Dimension 1     0.131666
4  Dimension 3     0.101799
5  Dimension 5     0.096488
6  Dimension 4     0.085606
7  Dimension 6     0.059822
```

MDS

[34]:
```
mds = MDS(n_components=8, random_state=RANDOM_STATE)
matrix_with_mds_embeddings_data = mds.fit_transform(DATA)
mtx1, mtx2, disparity = procrustes(matrix_with_human_data,␣
 ↪matrix_with_mds_embeddings_data)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

mds_result = {
        'Embedding': 'MDS',
        'Disparity': disparity,
```

```
        'Correlation': stat,
        'P_Value': p_value
}

procrustes_analysis.append(mds_result)

print('Embedding: MDS')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: MDS
Disparity: 0.9888007069481013
Correlation: 0.1058267123740448
P_value: 1.2498480972715074e-08
```

[35]:
```
mds_corr_table = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  mds_corr_table.loc[len(mds_corr_table)] = curr_row

mds_corr_table = mds_corr_table.sort_values(by = ['Correlation'], ascending =␣
↪False).reset_index(drop = True)

print(mds_corr_table)
```

```
     Dimension  Correlation
0  Dimension 1     0.181589
1  Dimension 5     0.120547
2  Dimension 7     0.118565
3  Dimension 2     0.104490
4  Dimension 8     0.096400
5  Dimension 4     0.087505
6  Dimension 3     0.080163
7  Dimension 6     0.056289
```

[36]:
```
human_comparison_df = pd.DataFrame(procrustes_analysis).
↪sort_values(by='Disparity')
display(human_comparison_df)
```

```
  Embedding  Disparity  Correlation       P_Value
1       LLE   0.986907     0.114426  7.353328e-10
2       MDS   0.988801     0.105827  1.249848e-08
0       PCA   0.990307     0.098455  1.195346e-07
```

Answer

1. The Procrustes analysis has been done over the PCA, MDS and LLE transformations. Out of which, the LLE has the least disparity meaning that it is the most similar one out of the

three to the human data.

2. The 3 diparity values are very similar to each other but far from the ideal 0(no disprity) value.
3. The correlation for each dimension for the 3 transformations has been tabulated in the above cells.

Question 5 Cluster the 360 images using K-Means

[37]:
```
pca = PCA(n_components=.90, random_state=RANDOM_STATE)
matrix_with_pca_embeddings_data = pca.fit_transform(DATA)
kmeans = KMeans(n_clusters=3, random_state=RANDOM_STATE, n_init="auto")
kmeans.fit_predict(matrix_with_pca_embeddings_data)
silhouette_avg = silhouette_score(DATA, kmeans.labels_)
print(silhouette_avg)
```

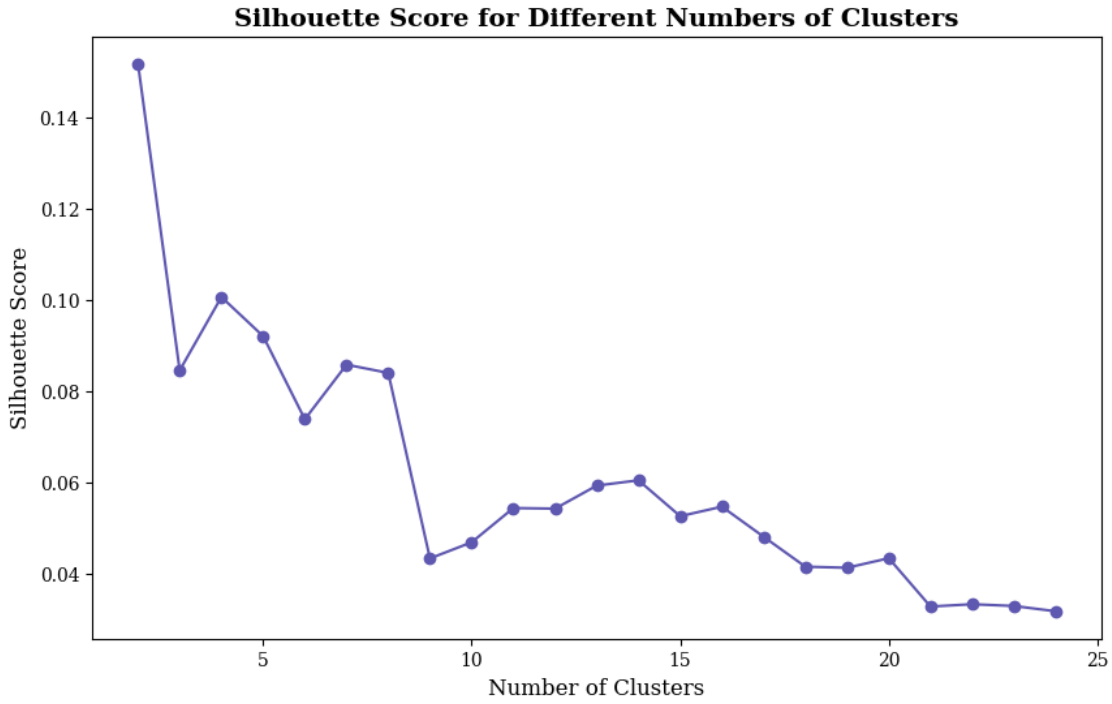
```
0.07358794064774253
```

[38]:
```
stat, p_value = pearsonr(encoded_labels, kmeans.labels_)
print(stat)
```

```
0.12171630256679136
```

[39]:
```
silhouette_scores = []
cluster_range = range(2, 25)
for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=RANDOM_STATE,␣
 ↪n_init="auto")
    cluster_labels = kmeans.fit_predict(matrix_with_pca_embeddings_data)
    score = silhouette_score(matrix_with_pca_embeddings_data, cluster_labels)
    silhouette_scores.append(score)
```

[40]:
```
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, silhouette_scores, marker='o', color=line_color)
plt.title("Silhouette Score for Different Numbers of Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.show()
```



Optimal number of clusters from the graph: 6

[41]:
```
pca = PCA(n_components=.90, random_state=RANDOM_STATE)
matrix_with_pca_embeddings_data = pca.fit_transform(DATA)
kmeans = KMeans(n_clusters=3, random_state=RANDOM_STATE, n_init="auto")
kmeans.fit_predict(matrix_with_pca_embeddings_data)

kmeans.labels_
```

[41]:
```
array([1, 1, 1, 2, 2, 1, 1, 2, 1, 2, 0, 2, 0, 2, 0, 2, 1, 2, 2, 0, 1, 0,
       0, 2, 0, 1, 0, 1, 2, 2, 2, 1, 2, 2, 2, 0, 1, 0, 0, 2, 0, 0, 1, 1,
       2, 2, 1, 2, 2, 1, 2, 0, 2, 1, 1, 2, 1, 0, 2, 0, 2, 0, 2, 1, 0, 2,
       2, 2, 2, 1, 2, 0, 2, 2, 2, 1, 0, 1, 2, 1, 2, 2, 2, 2, 2, 1, 0, 2,
       0, 0, 1, 1, 2, 2, 0, 1, 1, 1, 2, 1, 1, 0, 0, 0, 2, 2, 2, 2, 1, 1,
       1, 2, 0, 0, 1, 2, 2, 0, 2, 0, 2, 2, 0, 1, 2, 0, 0, 2, 1, 0, 2, 0,
       2, 2, 1, 2, 2, 0, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 0, 0,
       0, 2, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 1, 2, 1, 0, 2, 2, 0, 0, 2, 0,
       2, 0, 2, 1, 2, 1, 1, 0, 2, 1, 2, 2, 1, 1, 1, 2, 1, 2, 1, 0, 1, 2,
       2, 2, 2, 2, 2, 0, 1, 2, 0, 2, 2, 0, 2, 2, 2, 2, 0, 1, 2, 0, 0, 1,
       0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 2, 1, 2, 2, 1, 1, 2, 1, 0, 2,
       0, 1, 2, 0, 2, 2, 2, 0, 2, 1, 1, 1, 1, 1, 2, 1, 1, 0, 2, 0, 1, 0,
       0, 2, 1, 2, 0, 0, 2, 2, 0, 1, 1, 2, 1, 1, 1, 0, 2, 2, 1, 2, 1, 1,
       2, 1, 2, 0, 2, 0, 0, 2, 0, 0, 0, 1, 1, 2, 0, 2, 1, 0, 0, 2, 2, 1,
       1, 0, 0, 2, 0, 1, 1, 2, 1, 1, 0, 0, 1, 0, 0, 1, 0, 2, 2, 1, 2, 2,
```

```
       1, 2, 2, 2, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1,
       1, 1, 2, 2, 2, 2, 0, 1], dtype=int32)
```

[42]:
```
acc_score = accuracy_score(encoded_labels, kmeans.labels_)
print("Accuracy of KMeans clustering with PCA and number of clusters as 3: {0}".
  ↪format(acc_score))
```

```
Accuracy of KMeans clustering with PCA and number of clusters as 3:
0.34444444444444444
```

Answer 1. PCA has been used to reduce the number of dimensions preserving 90% of the variance.

2. Silhouette Score has been used to determine the number of optimum clusters. The analysis reveals that 6 is the number of optimal clusters in the dataset.

3. Since there are only 3 categories of rocks, KMeans clustering has been performed using number of clusters as 3. The accuracy score thus obtained is 0.34444444444444444.

Question 6

Cluster the 360 images using EM

[43]:
```
pca = PCA(n_components=.90, random_state=RANDOM_STATE)
matrix_with_pca_embeddings_data = pca.fit_transform(DATA)
gm = GaussianMixture(n_components=3, random_state=RANDOM_STATE)

predicted = gm.fit_predict(matrix_with_pca_embeddings_data)
```

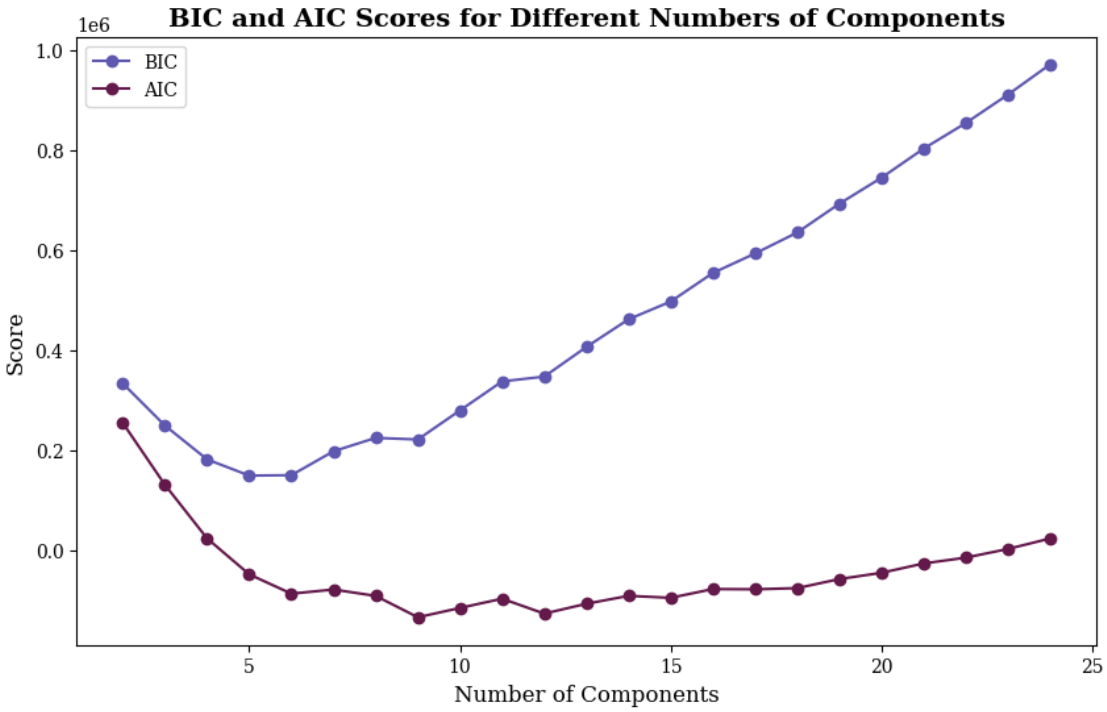
[44]:
```
stat, p_value = pearsonr(encoded_labels, predicted)
print(stat, p_value)
```

```
0.12171630256679136 0.02088994886753403
```

[45]:
```
bic_scores = []
aic_scores = []

for n_components in cluster_range:
    gm = GaussianMixture(n_components=n_components, random_state=RANDOM_STATE)
    gm.fit(matrix_with_pca_embeddings_data)
    bic_scores.append(gm.bic(matrix_with_pca_embeddings_data))
    aic_scores.append(gm.aic(matrix_with_pca_embeddings_data))
```

[46]:
```
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, bic_scores, label='BIC', marker='o', color=COLOR_MAP(.
  ↪3))
plt.plot(cluster_range, aic_scores, label='AIC', marker='o', color=COLOR_MAP(.
  ↪6))
plt.title("BIC and AIC Scores for Different Numbers of Components")
plt.xlabel("Number of Components")
```

```
plt.ylabel("Score")
plt.legend()
plt.show()
```



Optimal number of clusters from the graph: 6

[47]:
```
pca = PCA(n_components=.90, random_state=RANDOM_STATE)
matrix_with_pca_embeddings_data = pca.fit_transform(DATA)
gm = GaussianMixture(n_components=3, random_state=RANDOM_STATE)

predicted = gm.fit_predict(matrix_with_pca_embeddings_data)
```

[48]:
```
predicted
```

[48]:
```
array([1, 1, 1, 2, 2, 1, 1, 2, 1, 2, 0, 2, 0, 2, 0, 2, 1, 2, 2, 0, 1, 0,
       0, 2, 0, 1, 0, 1, 2, 2, 2, 1, 2, 2, 2, 0, 1, 0, 0, 2, 0, 0, 1, 1,
       2, 2, 1, 2, 2, 1, 2, 0, 2, 1, 1, 2, 1, 0, 2, 0, 2, 0, 2, 1, 0, 2,
       2, 2, 2, 1, 2, 0, 2, 2, 2, 1, 0, 1, 2, 1, 2, 2, 2, 2, 2, 1, 0, 2,
       0, 0, 1, 1, 2, 2, 0, 1, 1, 1, 2, 1, 1, 0, 0, 0, 2, 2, 2, 2, 1, 1,
       1, 2, 0, 0, 1, 2, 2, 0, 2, 0, 2, 2, 0, 1, 2, 0, 0, 2, 1, 0, 2, 0,
       2, 2, 1, 2, 2, 0, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 0, 0,
       0, 2, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 1, 2, 1, 0, 2, 2, 0, 0, 2, 0,
       2, 0, 2, 1, 2, 1, 1, 0, 2, 1, 2, 2, 1, 1, 1, 2, 1, 2, 1, 0, 1, 2,
       2, 2, 2, 2, 2, 0, 1, 2, 0, 2, 2, 0, 2, 2, 2, 2, 0, 1, 2, 0, 0, 1,
```

```
0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 2, 1, 2, 2, 1, 1, 2, 1, 0, 2,
0, 1, 2, 0, 2, 2, 2, 0, 2, 1, 1, 1, 1, 1, 2, 1, 1, 0, 2, 0, 1, 0,
0, 2, 1, 2, 0, 0, 2, 2, 0, 1, 1, 2, 1, 1, 1, 0, 2, 2, 1, 2, 1, 1,
2, 1, 2, 0, 2, 0, 0, 2, 0, 0, 0, 1, 1, 2, 0, 2, 1, 0, 0, 2, 2, 1,
1, 0, 0, 2, 0, 1, 1, 2, 1, 1, 0, 0, 1, 0, 0, 1, 0, 2, 2, 1, 2, 2,
1, 2, 2, 2, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1,
1, 1, 2, 2, 2, 2, 0, 1])
```

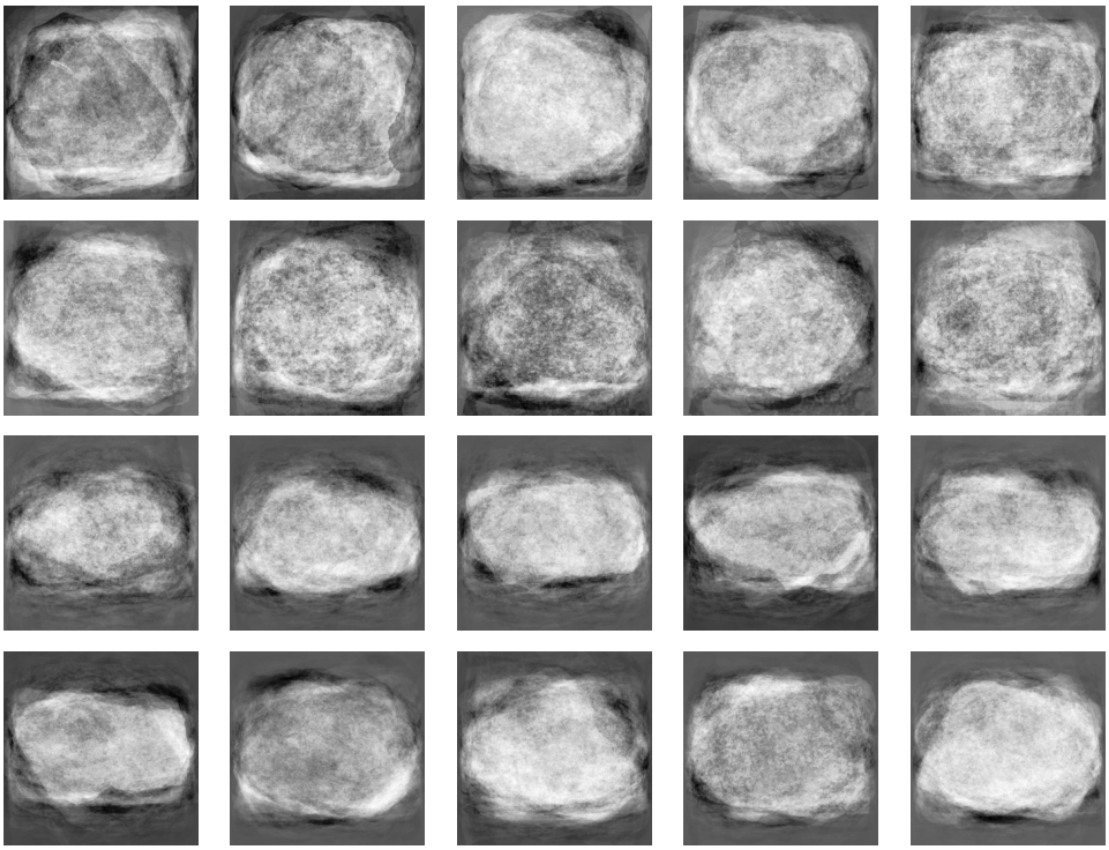
[49]:
```
acc_score = accuracy_score(encoded_labels, predicted)
print("Accuracy of EM clustering with PCA and number of clusters as 3: {0}".
 ↪format(acc_score))
```

```
Accuracy of EM clustering with PCA and number of clusters as 3:
0.34444444444444444
```

Sampling Images from the Gaussian Distributions sampling 20 random rocks

[50]:
```
gm = GaussianMixture(n_components=6, random_state=RANDOM_STATE)
gm.fit(matrix_with_pca_embeddings_data)

samples, _ = gm.sample(n_samples=20)
original_space_samples = pca.inverse_transform(samples)

fig, axes = plt.subplots(4, 5, figsize=(10, 8))
for i, ax in enumerate(axes.ravel()):
    ax.imshow(np.array(original_space_samples[i, : ]).reshape((256, 256)), cmap␣
 ↪= 'Greys')
    ax.axis('off')
plt.suptitle("Gaussian Mixture Sampled Rocks")
plt.tight_layout()
plt.show()
```

Gaussian Mixture Sampled Rocks



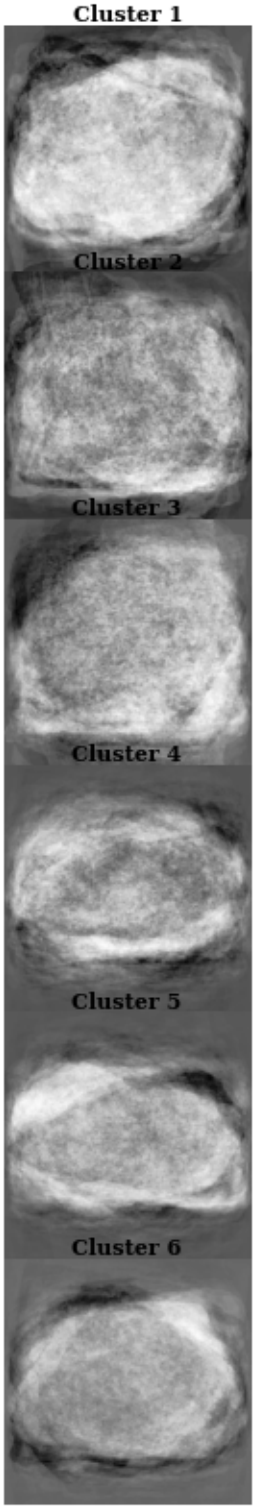
Sampling 2 rocks from each cluster

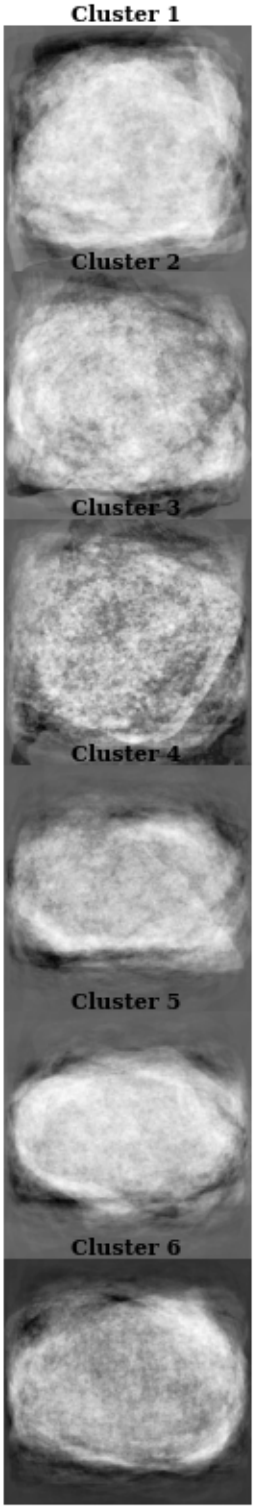
[51]:
```
n_samples_per_cluster = 2
original_space_samples = []

for cluster_idx in range(gm.n_components):

    mean = gm.means_[cluster_idx]
    cov = gm.covariances_[cluster_idx]
    samples_from_cluster = np.random.multivariate_normal(mean, cov,␣
 ↪n_samples_per_cluster)

    original_space_samples.extend(pca.inverse_transform(samples_from_cluster))

original_space_samples = np.array(original_space_samples)
```

[52]:
```
fig, axes = plt.subplots(gm.n_components, n_samples_per_cluster, figsize=(10,␣
 ↪10))
for i, ax in enumerate(axes.ravel()):
```

```
    cluster_label = i // n_samples_per_cluster
    ax.imshow(np.array(original_space_samples[i, :]).reshape((256, 256)),␣
 ↪cmap='Greys')
    ax.axis('off')
    ax.set_title(f"Cluster {cluster_label+1}", fontsize=8, pad=2)

plt.suptitle("Gaussian Mixture Sampled Rocks For Each Cluster")
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```

Gaussian Mixture Sampled Rocks For Each Cluster





Answer

1. PCA has been used to reduce the number of dimensions preserving 90% of the variance.
2. AIC and BIC have been used to determine the number of optimum clusters. The analysis reveals that 6 is the number of optimal clusters in the dataset, same as Slihoutte score.
3. Since there are only 3 categories of rocks, EM clustering has been performed using number of clusters as 3. The accuracy score thus obtained is 0.34444444444444444.
4. 20 random rock images have been generated and visualized in the baove cells.

Question 7

[53]:
```
# !pip install torchinfo
```

[54]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, TensorDataset
from torchinfo import summary
```

[55]:
```
VALIDATION_DATA, VALIDATION_LABELS = load_images_to_dataframe(VALIDATION_FOLDER)

rock_mapping = {"Igneous": 0, "Metamorphic": 1, "Sedimentary": 2}
val_encoded_labels = VALIDATION_LABELS.map(rock_mapping)
```

Converting Data

[56]:
```
features = DATA.values
val_features = VALIDATION_DATA.values

print(len(features))
print(len(encoded_labels))

print(len(val_features))
print(len(val_encoded_labels))
```

```
360
360
120
120
```

[57]:
```
batch_size = 32
```

[58]:
```
features_tensor = torch.tensor(features, dtype=torch.float32)
labels_tensor = torch.tensor(encoded_labels, dtype=torch.long)
```

```
train_dataset = TensorDataset(features_tensor, labels_tensor)
train_dataloader = DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
```

[59]:
```
val_features_tensor = torch.tensor(val_features, dtype=torch.float32)
val_labels_tensor = torch.tensor(val_encoded_labels, dtype=torch.long)

val_dataset = TensorDataset(val_features_tensor, val_labels_tensor)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=True)
```

[60]:
```
print(f"Labels: {val_labels_tensor.unique()}")
print(f"Labels: {labels_tensor.unique()}")
```

```
Labels: tensor([0, 1, 2])
Labels: tensor([0, 1, 2])
```

[61]:
```
print(val_dataset[0])
```

```
(tensor([1., 1., 1.,  …, 1., 1., 1.]), tensor(1))
```

Building The Model

[62]:
```
class RockClassifier(nn.Module):
    def __init__(self):
        super(RockClassifier, self).__init__()

        # Fully connected layers to process the flattened input
        self.fc1 = nn.Linear(256 * 256, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 32)
        self.fc6 = nn.Linear(32, 16)
        self.fc7 = nn.Linear(16, 8)
        self.relu = nn.ReLU()
        self.output_layer = nn.Linear(8, 3)

    def forward(self, x):
        # Fully connected layers with ReLU
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = self.relu(self.fc5(x))
        x = self.relu(self.fc6(x))
        x = self.relu(self.fc7(x))
        x = self.output_layer(x)
```

```
        return x



def train_model(model, Training_DataLoader, device, optimizer, criterion):
    model.train()
    train_correct, train_total, train_loss = 0, 0, 0

    for images, labels in Training_DataLoader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        train_correct += (torch.argmax(outputs, dim=1) == labels).sum().item()
        train_total += labels.size(0)


    return train_correct/train_total, train_loss

def eval_model(model, Test_DataLoader, device, criterion):
    model.eval()
    test_correct, test_total, test_loss = 0, 0, 0

    with torch.no_grad():
        for images, labels in Test_DataLoader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            test_loss += criterion(outputs, labels).item()
            test_correct+= (torch.argmax(outputs, dim=1) == labels).sum().item()
            test_total += labels.size(0)

    return test_correct/test_total, test_loss
```

[63]:
```
model = RockClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

[63]:
```
RockClassifier(
  (fc1): Linear(in_features=65536, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
```

```
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (fc5): Linear(in_features=64, out_features=32, bias=True)
  (fc6): Linear(in_features=32, out_features=16, bias=True)
  (fc7): Linear(in_features=16, out_features=8, bias=True)
  (relu): ReLU()
  (output_layer): Linear(in_features=8, out_features=3, bias=True)
)
```

7.A Report Training Time

[64]:
```
%time train_model(model, train_dataloader, device, optimizer, criterion)
```

```
CPU times: user 954 ms, sys: 138 ms, total: 1.09 s
Wall time: 710 ms
```

[64]:
```
(0.3333333333333333, 13.326364636421204)
```

7.B Plot training and validation lost as a function of training epochs

[65]:
```
num_epochs = 100
```

[66]:
```
import time
# Lists to store metrics for each epoch
training_accuracy = []
validation_accuracy = []
training_loss = []
validation_loss = []
ist = time.time()

for epoch in range(num_epochs):
    st = time.time()
    print(f"Current Epoch: {epoch+1}")

    # Training
    train_accuracy, train_loss = train_model(model, train_dataloader, device,␣
 ↪optimizer, criterion)
    training_accuracy.append(train_accuracy)
    training_loss.append(train_loss)

    # Validation
    val_accuracy, val_loss = eval_model(model, val_dataloader, device,␣
 ↪criterion)
    validation_accuracy.append(val_accuracy)
    validation_loss.append(val_loss)
```

```
    print("Metrics: Train Accuracy = {0}, Train Loss = {1}, Validation Accuracy␣
 ↪= {2}, Validation Loss = {3}, Epoch Time = {4} seconds".
 ↪format(train_accuracy, train_loss, val_accuracy, val_loss, time.time() - st))

print("Average Time per Epoch: {0} seconds".format((time.time() - ist) /␣
 ↪num_epochs))
```

```
Current Epoch: 1
Metrics: Train Accuracy = 0.3333333333333333, Train Loss = 13.24045729637146,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.413454055786133,
Epoch Time = 0.394122838973999 seconds
Current Epoch: 2
Metrics: Train Accuracy = 0.3333333333333333, Train Loss = 13.213616251945496,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.406772613525391,
Epoch Time = 0.39191389083862305 seconds
Current Epoch: 3
Metrics: Train Accuracy = 0.30833333333333335, Train Loss = 13.208842635154724,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.402699828147888,
Epoch Time = 0.39893579483032227 seconds
Current Epoch: 4
Metrics: Train Accuracy = 0.35555555555555557, Train Loss = 13.140219569206238,
Validation Accuracy = 0.325, Validation Loss = 4.400127649307251, Epoch Time =
0.3898129463195801 seconds
Current Epoch: 5
Metrics: Train Accuracy = 0.31666666666666665, Train Loss = 13.149726867675781,
Validation Accuracy = 0.3, Validation Loss = 4.397853374481201, Epoch Time =
0.38872456550598145 seconds
Current Epoch: 6
Metrics: Train Accuracy = 0.3277777777777778, Train Loss = 13.140344738960266,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.402556777000427,
Epoch Time = 0.4013535976409912 seconds
Current Epoch: 7
Metrics: Train Accuracy = 0.33611111111111114, Train Loss = 13.119879841804504,
Validation Accuracy = 0.31666666666666665, Validation Loss = 4.405545711517334,
Epoch Time = 0.4044382572174072 seconds
Current Epoch: 8
Metrics: Train Accuracy = 0.375, Train Loss = 13.13178288936615, Validation
Accuracy = 0.3333333333333333, Validation Loss = 4.396512389183044, Epoch Time =
0.3919193744659424 seconds
Current Epoch: 9
Metrics: Train Accuracy = 0.34444444444444444, Train Loss = 13.138274669647217,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.3972978591918945,
Epoch Time = 0.39301562309265137 seconds
Current Epoch: 10
Metrics: Train Accuracy = 0.39444444444444443, Train Loss = 13.092850685119629,
Validation Accuracy = 0.36666666666666664, Validation Loss = 4.397776246070862,
Epoch Time = 0.4017372131347656 seconds
```

```
Current Epoch: 11
Metrics: Train Accuracy = 0.4305555555555556, Train Loss = 13.082475543022156,
Validation Accuracy = 0.31666666666666665, Validation Loss = 4.392743706703186,
Epoch Time = 0.3949239253997803 seconds
Current Epoch: 12
Metrics: Train Accuracy = 0.4027777777777778, Train Loss = 13.064759731292725,
Validation Accuracy = 0.325, Validation Loss = 4.414789319038391, Epoch Time =
0.4157586097717285 seconds
Current Epoch: 13
Metrics: Train Accuracy = 0.41944444444444445, Train Loss = 13.028628468513489,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.402665972709656,
Epoch Time = 0.41081666946411133 seconds
Current Epoch: 14
Metrics: Train Accuracy = 0.4027777777777778, Train Loss = 13.03770911693573,
Validation Accuracy = 0.30833333333333335, Validation Loss = 4.394160985946655,
Epoch Time = 0.38272881507873535 seconds
Current Epoch: 15
Metrics: Train Accuracy = 0.41944444444444445, Train Loss = 12.923834681510925,
Validation Accuracy = 0.35833333333333334, Validation Loss = 4.3947800397872925,
Epoch Time = 0.4157376289367676 seconds
Current Epoch: 16
Metrics: Train Accuracy = 0.42777777777777776, Train Loss = 12.986361145973206,
Validation Accuracy = 0.35, Validation Loss = 4.404331803321838, Epoch Time =
0.39118432998657227 seconds
Current Epoch: 17
Metrics: Train Accuracy = 0.48333333333333334, Train Loss = 12.811157822608948,
Validation Accuracy = 0.35833333333333334, Validation Loss = 4.4002463817596436,
Epoch Time = 0.3942301273345947 seconds
Current Epoch: 18
Metrics: Train Accuracy = 0.4166666666666667, Train Loss = 12.828574538230896,
Validation Accuracy = 0.325, Validation Loss = 4.404513120651245, Epoch Time =
0.39866113662719727 seconds
Current Epoch: 19
Metrics: Train Accuracy = 0.44166666666666665, Train Loss = 12.810690522193909,
Validation Accuracy = 0.35, Validation Loss = 4.406266689300537, Epoch Time =
0.39885973930358887 seconds
Current Epoch: 20
Metrics: Train Accuracy = 0.4388888888888889, Train Loss = 12.733255863189697,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.4178279638290405,
Epoch Time = 0.39604759216308594 seconds
Current Epoch: 21
Metrics: Train Accuracy = 0.4638888888888889, Train Loss = 12.576424658298492,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.429516792297363,
Epoch Time = 0.40858912467956543 seconds
Current Epoch: 22
Metrics: Train Accuracy = 0.5055555555555555, Train Loss = 12.644680082798004,
Validation Accuracy = 0.36666666666666664, Validation Loss = 4.422458529472351,
Epoch Time = 0.38481593132019043 seconds
```

```
Current Epoch: 23
Metrics: Train Accuracy = 0.4444444444444444, Train Loss = 12.403293669223785,
Validation Accuracy = 0.35833333333333334, Validation Loss = 4.475040674209595,
Epoch Time = 0.3945178985595703 seconds
Current Epoch: 24
Metrics: Train Accuracy = 0.5194444444444445, Train Loss = 12.263875722885132,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.556506991386414,
Epoch Time = 0.4087216854095459 seconds
Current Epoch: 25
Metrics: Train Accuracy = 0.4083333333333333, Train Loss = 12.969040393829346,
Validation Accuracy = 0.325, Validation Loss = 4.56925642490387, Epoch Time =
0.3969430923461914 seconds
Current Epoch: 26
Metrics: Train Accuracy = 0.40555555555555556, Train Loss = 12.806936264038086,
Validation Accuracy = 0.325, Validation Loss = 4.537645697593689, Epoch Time =
0.38912057876586914 seconds
Current Epoch: 27
Metrics: Train Accuracy = 0.4638888888888889, Train Loss = 12.485807597637177,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.47367787361145,
Epoch Time = 0.4040396213531494 seconds
Current Epoch: 28
Metrics: Train Accuracy = 0.43333333333333335, Train Loss = 12.669463038444519,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.53617525100708,
Epoch Time = 0.39749836921691895 seconds
Current Epoch: 29
Metrics: Train Accuracy = 0.5111111111111111, Train Loss = 12.282554686069489,
Validation Accuracy = 0.325, Validation Loss = 4.515152454376221, Epoch Time =
0.3981893062591553 seconds
Current Epoch: 30
Metrics: Train Accuracy = 0.5277777777777778, Train Loss = 12.184265315532684,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.528958439826965,
Epoch Time = 0.3951263427734375 seconds
Current Epoch: 31
Metrics: Train Accuracy = 0.5, Train Loss = 11.998605608940125, Validation
Accuracy = 0.3333333333333333, Validation Loss = 4.580593466758728, Epoch Time =
0.4066498279571533 seconds
Current Epoch: 32
Metrics: Train Accuracy = 0.5194444444444445, Train Loss = 12.115875780582428,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.69326376914978,
Epoch Time = 0.3882894515991211 seconds
Current Epoch: 33
Metrics: Train Accuracy = 0.5333333333333333, Train Loss = 12.144153118133545,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.637881875038147,
Epoch Time = 0.39042186737060547 seconds
Current Epoch: 34
Metrics: Train Accuracy = 0.525, Train Loss = 11.726536452770233, Validation
Accuracy = 0.375, Validation Loss = 4.596679091453552, Epoch Time =
0.3942427635192871 seconds
```

```
Current Epoch: 35
Metrics: Train Accuracy = 0.5333333333333333, Train Loss = 11.644373774528503,
Validation Accuracy = 0.31666666666666665, Validation Loss = 4.691516399383545,
Epoch Time = 0.39583420753479004 seconds
Current Epoch: 36
Metrics: Train Accuracy = 0.4222222222222222, Train Loss = 13.164454579353333,
Validation Accuracy = 0.30833333333333335, Validation Loss = 4.61923623085022,
Epoch Time = 0.41504883766174316 seconds
Current Epoch: 37
Metrics: Train Accuracy = 0.4861111111111111, Train Loss = 12.441640973091125,
Validation Accuracy = 0.31666666666666665, Validation Loss = 4.57106351852417,
Epoch Time = 0.39058375358581543 seconds
Current Epoch: 38
Metrics: Train Accuracy = 0.5361111111111111, Train Loss = 12.017323911190033,
Validation Accuracy = 0.31666666666666665, Validation Loss = 4.693928003311157,
Epoch Time = 0.40061068534851074 seconds
Current Epoch: 39
Metrics: Train Accuracy = 0.5694444444444444, Train Loss = 11.386253356933594,
Validation Accuracy = 0.3, Validation Loss = 4.8477559089660645, Epoch Time =
0.3842024803161621 seconds
Current Epoch: 40
Metrics: Train Accuracy = 0.5833333333333334, Train Loss = 11.125294983386993,
Validation Accuracy = 0.30833333333333335, Validation Loss = 5.017873644828796,
Epoch Time = 0.37627673149108887 seconds
Current Epoch: 41
Metrics: Train Accuracy = 0.5722222222222222, Train Loss = 11.408421814441681,
Validation Accuracy = 0.31666666666666665, Validation Loss = 5.0207719802856445,
Epoch Time = 0.403322696685791 seconds
Current Epoch: 42
Metrics: Train Accuracy = 0.5944444444444444, Train Loss = 11.309852421283722,
Validation Accuracy = 0.35833333333333334, Validation Loss = 4.77627694606781,
Epoch Time = 0.40428781509399414 seconds
Current Epoch: 43
Metrics: Train Accuracy = 0.5722222222222222, Train Loss = 10.875890672206879,
Validation Accuracy = 0.275, Validation Loss = 4.8443697690963745, Epoch Time =
0.39489006996154785 seconds
Current Epoch: 44
Metrics: Train Accuracy = 0.4111111111111111, Train Loss = 13.7169508934021,
Validation Accuracy = 0.30833333333333335, Validation Loss = 4.722595810890198,
Epoch Time = 0.4069385528564453 seconds
Current Epoch: 45
Metrics: Train Accuracy = 0.5416666666666666, Train Loss = 12.034357190132141,
Validation Accuracy = 0.3, Validation Loss = 4.664357781410217, Epoch Time =
0.3934953212738037 seconds
Current Epoch: 46
Metrics: Train Accuracy = 0.5222222222222223, Train Loss = 11.34806489944458,
Validation Accuracy = 0.30833333333333335, Validation Loss = 4.562563538551331,
Epoch Time = 0.39121532440185547 seconds
```

```
Current Epoch: 47
Metrics: Train Accuracy = 0.6055555555555555, Train Loss = 10.779706597328186,
Validation Accuracy = 0.3, Validation Loss = 4.70246422290802, Epoch Time =
0.39845943450927734 seconds
Current Epoch: 48
Metrics: Train Accuracy = 0.5416666666666666, Train Loss = 11.016088008880615,
Validation Accuracy = 0.3, Validation Loss = 5.273488759994507, Epoch Time =
0.4016449451446533 seconds
Current Epoch: 49
Metrics: Train Accuracy = 0.49722222222222223, Train Loss = 12.002389311790466,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.723020493984222,
Epoch Time = 0.3813304901123047 seconds
Current Epoch: 50
Metrics: Train Accuracy = 0.6027777777777777, Train Loss = 10.48162704706192,
Validation Accuracy = 0.325, Validation Loss = 4.926965832710266, Epoch Time =
0.40186333656311035 seconds
Current Epoch: 51
Metrics: Train Accuracy = 0.6222222222222222, Train Loss = 10.698125839233398,
Validation Accuracy = 0.325, Validation Loss = 5.24179470539093, Epoch Time =
0.39986443519592285 seconds
Current Epoch: 52
Metrics: Train Accuracy = 0.6027777777777777, Train Loss = 10.333012998104095,
Validation Accuracy = 0.3416666666666667, Validation Loss = 4.841925144195557,
Epoch Time = 0.39484238624572754 seconds
Current Epoch: 53
Metrics: Train Accuracy = 0.6416666666666667, Train Loss = 10.184981286525726,
Validation Accuracy = 0.3166666666666665, Validation Loss = 5.278070688247681,
Epoch Time = 0.3926818370819092 seconds
Current Epoch: 54
Metrics: Train Accuracy = 0.6277777777777778, Train Loss = 10.238701403141022,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.962779760360718,
Epoch Time = 0.4013514518737793 seconds
Current Epoch: 55
Metrics: Train Accuracy = 0.6083333333333333, Train Loss = 9.811877012252808,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.958821535110474,
Epoch Time = 0.40633678436279297 seconds
Current Epoch: 56
Metrics: Train Accuracy = 0.6583333333333333, Train Loss = 9.466301798820496,
Validation Accuracy = 0.325, Validation Loss = 5.010193943977356, Epoch Time =
0.39376163482666016 seconds
Current Epoch: 57
Metrics: Train Accuracy = 0.6361111111111111, Train Loss = 10.069628953933716,
Validation Accuracy = 0.3166666666666665, Validation Loss = 5.985689163208008,
Epoch Time = 0.4033777713775635 seconds
Current Epoch: 58
Metrics: Train Accuracy = 0.5888888888888889, Train Loss = 10.2106214761734,
Validation Accuracy = 0.3333333333333333, Validation Loss = 5.204540014266968,
Epoch Time = 0.40314292907714844 seconds
```

```
Current Epoch: 59
Metrics: Train Accuracy = 0.7083333333333334, Train Loss = 9.18954187631607,
Validation Accuracy = 0.325, Validation Loss = 5.078015923500061, Epoch Time =
0.40320277214050293 seconds
Current Epoch: 60
Metrics: Train Accuracy = 0.7222222222222222, Train Loss = 8.790128231048584,
Validation Accuracy = 0.31666666666666665, Validation Loss = 5.719271421432495,
Epoch Time = 0.40202832221984863 seconds
Current Epoch: 61
Metrics: Train Accuracy = 0.7194444444444444, Train Loss = 8.637547850608826,
Validation Accuracy = 0.35833333333333334, Validation Loss = 5.127727508544922,
Epoch Time = 0.40699028968811035 seconds
Current Epoch: 62
Metrics: Train Accuracy = 0.7222222222222222, Train Loss = 8.531707048416138,
Validation Accuracy = 0.275, Validation Loss = 6.3301273584365845, Epoch Time =
0.3837754726409912 seconds
Current Epoch: 63
Metrics: Train Accuracy = 0.6638888888888889, Train Loss = 9.313976287841797,
Validation Accuracy = 0.30833333333333335, Validation Loss = 6.044297575950623,
Epoch Time = 0.40772080421447754 seconds
Current Epoch: 64
Metrics: Train Accuracy = 0.625, Train Loss = 9.601197600364685, Validation
Accuracy = 0.3333333333333333, Validation Loss = 5.3907318115234375, Epoch Time
= 0.4080955982208252 seconds
Current Epoch: 65
Metrics: Train Accuracy = 0.5666666666666667, Train Loss = 10.990935772657394,
Validation Accuracy = 0.3, Validation Loss = 5.354303598403931, Epoch Time =
0.3959498405456543 seconds
Current Epoch: 66
Metrics: Train Accuracy = 0.5638888888888889, Train Loss = 11.300332903862,
Validation Accuracy = 0.4083333333333333, Validation Loss = 4.789405703544617,
Epoch Time = 0.39694786071777344 seconds
Current Epoch: 67
Metrics: Train Accuracy = 0.6833333333333333, Train Loss = 8.998388230800629,
Validation Accuracy = 0.3416666666666667, Validation Loss = 5.353398561477661,
Epoch Time = 0.3783743381500244 seconds
Current Epoch: 68
Metrics: Train Accuracy = 0.7361111111111112, Train Loss = 8.22425365447998,
Validation Accuracy = 0.3333333333333333, Validation Loss = 5.223431706428528,
Epoch Time = 0.38060665130615234 seconds
Current Epoch: 69
Metrics: Train Accuracy = 0.7888888888888889, Train Loss = 7.7543394565582275,
Validation Accuracy = 0.38333333333333336, Validation Loss = 5.113342046737671,
Epoch Time = 0.3965950012207031 seconds
Current Epoch: 70
Metrics: Train Accuracy = 0.7472222222222222, Train Loss = 8.157763481140137,
Validation Accuracy = 0.3333333333333333, Validation Loss = 6.125273585319519,
Epoch Time = 0.3908860683441162 seconds
```

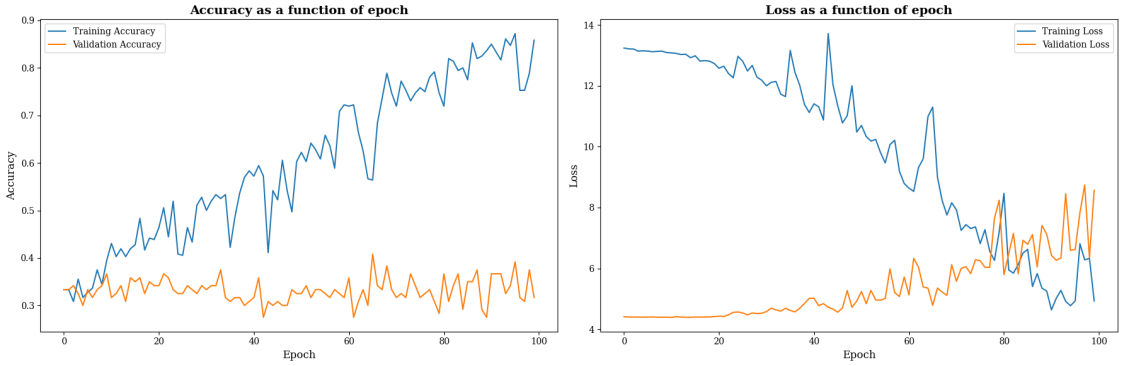
```
Current Epoch: 71
Metrics: Train Accuracy = 0.7194444444444444, Train Loss = 7.921670198440552,
Validation Accuracy = 0.31666666666666665, Validation Loss = 5.577584862709045,
Epoch Time = 0.39774012565612793 seconds
Current Epoch: 72
Metrics: Train Accuracy = 0.7722222222222223, Train Loss = 7.249794155359268,
Validation Accuracy = 0.325, Validation Loss = 6.002149343490601, Epoch Time =
0.3878934383392334 seconds
Current Epoch: 73
Metrics: Train Accuracy = 0.7527777777777778, Train Loss = 7.439304322004318,
Validation Accuracy = 0.31666666666666665, Validation Loss = 6.055707335472107,
Epoch Time = 0.39810872077941895 seconds
Current Epoch: 74
Metrics: Train Accuracy = 0.7305555555555555, Train Loss = 7.311917006969452,
Validation Accuracy = 0.36666666666666664, Validation Loss = 5.828109502792358,
Epoch Time = 0.3904988765716553 seconds
Current Epoch: 75
Metrics: Train Accuracy = 0.7472222222222222, Train Loss = 7.366958856582642,
Validation Accuracy = 0.3416666666666667, Validation Loss = 6.284390807151794,
Epoch Time = 0.39768528938293457 seconds
Current Epoch: 76
Metrics: Train Accuracy = 0.7583333333333333, Train Loss = 6.817408442497253,
Validation Accuracy = 0.31666666666666665, Validation Loss = 6.251220464706421,
Epoch Time = 0.40616321563720703 seconds
Current Epoch: 77
Metrics: Train Accuracy = 0.75, Train Loss = 7.273036301136017, Validation
Accuracy = 0.325, Validation Loss = 6.039660930633545, Epoch Time =
0.38643598556518555 seconds
Current Epoch: 78
Metrics: Train Accuracy = 0.7805555555555556, Train Loss = 6.566931277513504,
Validation Accuracy = 0.3333333333333333, Validation Loss = 6.032018542289734,
Epoch Time = 0.39223623275756836 seconds
Current Epoch: 79
Metrics: Train Accuracy = 0.7916666666666666, Train Loss = 6.263396203517914,
Validation Accuracy = 0.30833333333333335, Validation Loss = 7.645429015159607,
Epoch Time = 0.4016604423522949 seconds
Current Epoch: 80
Metrics: Train Accuracy = 0.7472222222222222, Train Loss = 7.211252719163895,
Validation Accuracy = 0.2833333333333333, Validation Loss = 8.24140191078186,
Epoch Time = 0.38889479637145996 seconds
Current Epoch: 81
Metrics: Train Accuracy = 0.7194444444444444, Train Loss = 8.467244684696198,
Validation Accuracy = 0.36666666666666664, Validation Loss = 5.796228289604187,
Epoch Time = 0.3824155330657959 seconds
Current Epoch: 82
Metrics: Train Accuracy = 0.8194444444444444, Train Loss = 5.952313601970673,
Validation Accuracy = 0.30833333333333335, Validation Loss = 6.474996089935303,
Epoch Time = 0.39122533798217773 seconds
```

```
Current Epoch: 83
Metrics: Train Accuracy = 0.8138888888888889, Train Loss = 5.843604147434235,
Validation Accuracy = 0.3416666666666667, Validation Loss = 7.153464436531067,
Epoch Time = 0.3859717845916748 seconds
Current Epoch: 84
Metrics: Train Accuracy = 0.7944444444444444, Train Loss = 6.130005180835724,
Validation Accuracy = 0.36666666666666664, Validation Loss = 5.822260141372681,
Epoch Time = 0.3895869255065918 seconds
Current Epoch: 85
Metrics: Train Accuracy = 0.8, Train Loss = 6.50429031252861, Validation
Accuracy = 0.2916666666666667, Validation Loss = 6.923892021179199, Epoch Time =
0.401869535446167 seconds
Current Epoch: 86
Metrics: Train Accuracy = 0.775, Train Loss = 6.6266820430755615, Validation
Accuracy = 0.35, Validation Loss = 6.79612672328949, Epoch Time =
0.3778526782989502 seconds
Current Epoch: 87
Metrics: Train Accuracy = 0.8527777777777777, Train Loss = 5.404515862464905,
Validation Accuracy = 0.35, Validation Loss = 7.111339330673218, Epoch Time =
0.3995659351348877 seconds
Current Epoch: 88
Metrics: Train Accuracy = 0.8194444444444444, Train Loss = 5.830965518951416,
Validation Accuracy = 0.375, Validation Loss = 6.049666404724121, Epoch Time =
0.40381312370300293 seconds
Current Epoch: 89
Metrics: Train Accuracy = 0.825, Train Loss = 5.349885314702988, Validation
Accuracy = 0.2916666666666667, Validation Loss = 7.414190292358398, Epoch Time =
0.3991680145263672 seconds
Current Epoch: 90
Metrics: Train Accuracy = 0.8361111111111111, Train Loss = 5.257259160280228,
Validation Accuracy = 0.275, Validation Loss = 7.12529718875885, Epoch Time =
0.4060845375061035 seconds
Current Epoch: 91
Metrics: Train Accuracy = 0.85, Train Loss = 4.6364719569683075, Validation
Accuracy = 0.36666666666666664, Validation Loss = 6.429833889007568, Epoch Time
= 0.39509010314941406 seconds
Current Epoch: 92
Metrics: Train Accuracy = 0.8333333333333334, Train Loss = 5.017831772565842,
Validation Accuracy = 0.36666666666666664, Validation Loss = 6.269439697265625,
Epoch Time = 0.41374826431274414 seconds
Current Epoch: 93
Metrics: Train Accuracy = 0.8166666666666667, Train Loss = 5.280003592371941,
Validation Accuracy = 0.36666666666666664, Validation Loss = 6.342032074928284,
Epoch Time = 0.4104905128479004 seconds
Current Epoch: 94
Metrics: Train Accuracy = 0.8611111111111112, Train Loss = 4.916973203420639,
Validation Accuracy = 0.325, Validation Loss = 8.45612120628357, Epoch Time =
0.3907012939453125 seconds
```

```
Current Epoch: 95
Metrics: Train Accuracy = 0.8472222222222222, Train Loss = 4.77446386218071,
Validation Accuracy = 0.3416666666666667, Validation Loss = 6.5946608781814575,
Epoch Time = 0.3990931510925293 seconds
Current Epoch: 96
Metrics: Train Accuracy = 0.8722222222222222, Train Loss = 4.932713091373444,
Validation Accuracy = 0.39166666666666666, Validation Loss = 6.623041391372681,
Epoch Time = 0.40637660026550293 seconds
Current Epoch: 97
Metrics: Train Accuracy = 0.7527777777777778, Train Loss = 6.8124256283044815,
Validation Accuracy = 0.31666666666666665, Validation Loss = 7.831965923309326,
Epoch Time = 0.39711570739746094 seconds
Current Epoch: 98
Metrics: Train Accuracy = 0.7527777777777778, Train Loss = 6.280791476368904,
Validation Accuracy = 0.30833333333333335, Validation Loss = 8.745145320892334,
Epoch Time = 0.40070414543151855 seconds
Current Epoch: 99
Metrics: Train Accuracy = 0.7888888888888889, Train Loss = 6.334947645664215,
Validation Accuracy = 0.375, Validation Loss = 6.350216627120972, Epoch Time =
0.3889896869659424 seconds
Current Epoch: 100
Metrics: Train Accuracy = 0.8583333333333333, Train Loss = 4.93022483587265,
Validation Accuracy = 0.31666666666666665, Validation Loss = 8.56767201423645,
Epoch Time = 0.39224720001220703 seconds
Average Time per Epoch: 0.3969054079055786 seconds
```

[67]:
```python
x = list(range(num_epochs))

fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot accuracy
axs[0].plot(x, training_accuracy, label='Training Accuracy')
axs[0].plot(x, validation_accuracy, label='Validation Accuracy')
axs[0].set_title('Accuracy as a function of epoch')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].legend()

# Plot loss
axs[1].plot(x, training_loss, label='Training Loss')
axs[1].plot(x, validation_loss, label='Validation Loss')
axs[1].set_title('Loss as a function of epoch')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Loss')
axs[1].legend()

# Adjust layout
```

```
plt.tight_layout()
plt.show()
```



7.C How many parameters does the network have? How many of those parameters are bias parameters?

Number of Weights and Biases per Layer

[68]:
```
bias_parameters = 0

for name, param in model.named_parameters():
    if "weight" in name:
        print(f"Layer: {name} | Weights: {len(param.data)}")
    elif "bias" in name:
        print(f"Layer: {name} | Biases: {len(param.data)}")
        bias_parameters += len(param.data)
```

```
Layer: fc1.weight | Weights: 512
Layer: fc1.bias | Biases: 512
Layer: fc2.weight | Weights: 256
Layer: fc2.bias | Biases: 256
Layer: fc3.weight | Weights: 128
Layer: fc3.bias | Biases: 128
Layer: fc4.weight | Weights: 64
Layer: fc4.bias | Biases: 64
Layer: fc5.weight | Weights: 32
Layer: fc5.bias | Biases: 32
Layer: fc6.weight | Weights: 16
Layer: fc6.bias | Biases: 16
Layer: fc7.weight | Weights: 8
Layer: fc7.bias | Biases: 8
Layer: output_layer.weight | Weights: 3
Layer: output_layer.bias | Biases: 3
```

[69]:
```
print("Total Number of Biases: {0}".format(bias_parameters))
```

```
Total Number of Biases: 1019
```

Neural Network Architecture

[70]:
```
summary(model, input_size=(batch_size, 256*256))
```

[70]:
```
==========================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
==========
RockClassifier                           [32, 3]                   --
 Linear: 1-1                             [32, 512]                 33,554,944
 ReLU: 1-2                               [32, 512]                 --
 Linear: 1-3                             [32, 256]                 131,328
 ReLU: 1-4                               [32, 256]                 --
 Linear: 1-5                             [32, 128]                 32,896
 ReLU: 1-6                               [32, 128]                 --
 Linear: 1-7                             [32, 64]                  8,256
 ReLU: 1-8                               [32, 64]                  --
 Linear: 1-9                             [32, 32]                  2,080
 ReLU: 1-10                              [32, 32]                  --
 Linear: 1-11                            [32, 16]                  528
 ReLU: 1-12                              [32, 16]                  --
 Linear: 1-13                            [32, 8]                   136
 ReLU: 1-14                              [32, 8]                   --
 Linear: 1-15                            [32, 3]                   27
==========================================================================
==========
Total params: 33,730,195
Trainable params: 33,730,195
Non-trainable params: 0
Total mult-adds (G): 1.08
==========================================================================
==========
Input size (MB): 8.39
Forward/backward pass size (MB): 0.26
Params size (MB): 134.92
Estimated Total Size (MB): 143.57
==========================================================================
==========
```

[71]:
```
print("Total number of parameters in the model: ", 33,730,195)
print("Total number of trainable parameters in the model: ", 33,730,195)
print("Total number of non-trainable parameters in the model: ", 0)
print("Total number of biases in the model: ", bias_parameters)
```

```
Total number of parameters in the model:  33 730 195
Total number of trainable parameters in the model:  33 730 195
Total number of non-trainable parameters in the model:  0
Total number of biases in the model:  1019
```

7.D

[72]:
```
for i in model.output_layer.weight:
    print(len(i))
```

```
8
8
8
```

[73]:
```
model
```

[73]:
```
RockClassifier(
  (fc1): Linear(in_features=65536, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (fc5): Linear(in_features=64, out_features=32, bias=True)
  (fc6): Linear(in_features=32, out_features=16, bias=True)
  (fc7): Linear(in_features=16, out_features=8, bias=True)
  (relu): ReLU()
  (output_layer): Linear(in_features=8, out_features=3, bias=True)
)
```

[74]:
```
torch.manual_seed(RANDOM_STATE)

all_outputs_train = []

model.eval()
for images, labels in train_dataloader:
    images = images.to(device)
    labels = labels.to(device)

    x = model.relu(model.fc1(images))
    x = model.relu(model.fc2(x))
    x = model.relu(model.fc3(x))
    x = model.relu(model.fc4(x))
    x = model.relu(model.fc5(x))
    x = model.relu(model.fc6(x))
    output_fc7 = model.relu(model.fc7(x))

    all_outputs_train.append(output_fc7.cpu())

all_outputs_train_combined = torch.cat(all_outputs_train, dim=0)
```

```
all_outputs_train_numpy = all_outputs_train_combined.detach().numpy()

print(all_outputs_train_numpy.shape)
```

```
(360, 8)
```

[75]:
```
matrix_with_human_data
```

[75]:
```
array([[-3.743, -1.204,  2.001, …, -1.992,  4.95 ,  1.695],
       [ 2.332,  1.625,  0.985, …,  0.093,  6.724,  0.708],
       [ 0.346,  1.49 , -3.795, …, -3.786,  0.706, -2.854],
       …,
       [-3.475, -3.431, -2.184, …, -2.265,  1.129, -1.201],
       [-0.051, -2.358,  1.994, …,  7.268, -0.593, -1.432],
       [ 1.134, -4.9  ,  0.983, …,  4.695,  0.624, -1.195]])
```

[76]:
```
all_outputs_train_numpy
```

[76]:
```
array([[0.       , 2.074276 , 2.5947733, …, 2.701891 , 7.348477 ,
        0.       ],
       [0.       , 2.8402746, 3.7883651, …, 3.446808 , 6.5583143,
        0.       ],
       [0.       , 5.9693327, 1.6653625, …, 7.3762884, 7.838596 ,
        0.       ],
       …,
       [0.       , 3.877427 , 3.7903612, …, 4.8101997, 8.080968 ,
        0.       ],
       [0.       , 3.7448452, 6.3102546, …, 4.226236 , 5.4096375,
        0.       ],
       [0.       , 2.361446 , 2.7314305, …, 3.0395548, 7.462207 ,
        0.       ]], dtype=float32)
```

[77]:
```
mtx1, mtx2, disparity = procrustes(matrix_with_human_data,␣
 ↪all_outputs_train_numpy)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

nn_train_result = {
        'Embedding': 'Neural Network Train',
        'Disparity': disparity,
        'Correlation': stat,
        'P_Value': p_value
}

procrustes_analysis.append(pca_result)
```

```
print('Embedding: Neural Network Train')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: Neural Network Train
Disparity: 0.9921825578267668
Correlation: 0.08841630038196205
P_value: 2.0124930187957532e-06
```

[78]:
```
nn_corr_table_train = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  nn_corr_table_train.loc[len(nn_corr_table_train)] = curr_row

nn_corr_table_train = nn_corr_table_train.sort_values(by = ['Correlation'],␣
 ↪ascending = False).reset_index(drop = True)

print(nn_corr_table_train)
```

```
     Dimension  Correlation
0  Dimension 2     0.123521
1  Dimension 5     0.118338
2  Dimension 4     0.117991
3  Dimension 6     0.086969
4  Dimension 7     0.071480
5  Dimension 8     0.071007
6  Dimension 3     0.065253
7  Dimension 1     0.049553
```

[79]:
```
torch.manual_seed(RANDOM_STATE)

all_outputs_valid = []

model.eval()
for images, labels in val_dataloader:
    images = images.to(device)
    labels = labels.to(device)

    x = model.relu(model.fc1(images))
    x = model.relu(model.fc2(x))
    x = model.relu(model.fc3(x))
    x = model.relu(model.fc4(x))
    x = model.relu(model.fc5(x))
    x = model.relu(model.fc6(x))
    output_fc7 = model.relu(model.fc7(x))
```

```
    all_outputs_valid.append(output_fc7.cpu())

all_outputs_valid_combined = torch.cat(all_outputs_valid, dim=0)

all_outputs_valid_numpy = all_outputs_valid_combined.detach().numpy()

print(all_outputs_valid_numpy.shape)
```

```
(120, 8)
```

[80]:
```
matrix_with_human_data_valid = np.loadtxt(HUMAN_PREDICTIVE_FEATURES_VALIDATION)

mtx1, mtx2, disparity = procrustes(matrix_with_human_data_valid,␣
 ↪all_outputs_valid_numpy)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

nn_train_result = {
        'Embedding': 'Neural Network Validation',
        'Disparity': disparity,
        'Correlation': stat,
        'P_Value': p_value
}

procrustes_analysis.append(pca_result)

print('Embedding: Neural Network Validation')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: Neural Network Validation
Disparity: 0.981745184034973
Correlation: 0.13511038437154615
P_value: 2.6684352771167693e-05
```

[81]:
```
nn_corr_table_valid = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  nn_corr_table_valid.loc[len(nn_corr_table_valid)] = curr_row

nn_corr_table_valid = nn_corr_table_valid.sort_values(by = ['Correlation'],␣
 ↪ascending = False).reset_index(drop = True)

print(nn_corr_table_valid)
```

```
     Dimension  Correlation
0  Dimension 6     0.202212
1  Dimension 5     0.175580
2  Dimension 7     0.173524
3  Dimension 3     0.153200
4  Dimension 8     0.135060
5  Dimension 4     0.113876
6  Dimension 2     0.100284
7  Dimension 1     0.047272
```

Trying out CNNs for a better classification

[82]:
```python
import torch
import torch.nn as nn

class RockClassifierCNN(nn.Module):
    def __init__(self):
        super(RockClassifierCNN, self).__init__()

        # Reversed order of convolutional layers
        self.conv1 = nn.Conv2d(1, 256, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1)
        self.conv5 = nn.Conv2d(32, 16, kernel_size=3, stride=1, padding=1)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.gap = nn.AdaptiveAvgPool2d(1)

        # Fully connected layers
        self.fc_intermediate = nn.Linear(16, 8)
        self.output_layer = nn.Linear(8, 3)

        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = self.pool(self.relu(self.conv4(x)))
        x = self.relu(self.conv5(x))

        x = self.gap(x)

        x = x.view(x.size(0), -1)
        x = self.relu(self.fc_intermediate(x))
        x = self.output_layer(x)
```

```
        return x

def train_model(model, Training_DataLoader, device, optimizer, criterion):
    model.train()
    train_correct, train_total, train_loss = 0, 0, 0

    for images, labels in Training_DataLoader:

        images = images.view(-1, 1, 256, 256).to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        train_correct += (torch.argmax(outputs, dim=1) == labels).sum().item()
        train_total += labels.size(0)


    return train_correct / train_total, train_loss


def eval_model(model, Test_DataLoader, device, criterion):
    model.eval()
    test_correct, test_total, test_loss = 0, 0, 0

    with torch.no_grad():
        for images, labels in Test_DataLoader:

            images = images.view(-1, 1, 256, 256).to(device)
            labels = labels.to(device)

            outputs = model(images)
            test_loss += criterion(outputs, labels).item()
            test_correct += (torch.argmax(outputs, dim=1) == labels).sum().
 ↪item()
            test_total += labels.size(0)

    return test_correct / test_total, test_loss


modelCNN = RockClassifierCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(modelCNN.parameters(), lr=0.0005)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
modelCNN.to(device)
```

[82]:
```
RockClassifierCNN(
  (conv1): Conv2d(1, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (gap): AdaptiveAvgPool2d(output_size=1)
  (fc_intermediate): Linear(in_features=16, out_features=8, bias=True)
  (output_layer): Linear(in_features=8, out_features=3, bias=True)
  (relu): ReLU()
)
```

[83]:
```
%time train_model(modelCNN, train_dataloader, device, optimizer, criterion)
```

```
CPU times: user 11.9 s, sys: 206 ms, total: 12.1 s
Wall time: 9.91 s
```

[83]:
```
(0.3333333333333333, 13.447508335113525)
```

[84]:
```
import time
num_epochs = 100

training_accuracy = []
validation_accuracy = []
training_loss = []
validation_loss = []
ist = time.time()

for epoch in range(num_epochs):
    st = time.time()
    print(f"Current Epoch: {epoch+1}")

    train_accuracy, train_loss = train_model(modelCNN, train_dataloader,␣
 ↪device, optimizer, criterion)
    training_accuracy.append(train_accuracy)
    training_loss.append(train_loss)

    val_accuracy, val_loss = eval_model(modelCNN, val_dataloader, device,␣
 ↪criterion)
    validation_accuracy.append(val_accuracy)
    validation_loss.append(val_loss)
```

```
    print("Metrics: Train Accuracy = {0}, Train Loss = {1}, Validation Accuracy␣
 ↪= {2}, Validation Loss = {3}, Epoch Time = {4} seconds".
 ↪format(train_accuracy, train_loss, val_accuracy, val_loss, time.time() - st))

print("Average Time per Epoch: {0} seconds".format((time.time() - ist) /␣
 ↪num_epochs))
```

```
Current Epoch: 1
Metrics: Train Accuracy = 0.3333333333333333, Train Loss = 13.305363774299622,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.416323781013489,
Epoch Time = 10.234906911849976 seconds
Current Epoch: 2
Metrics: Train Accuracy = 0.33055555555555555, Train Loss = 13.253665566444397,
Validation Accuracy = 0.38333333333333336, Validation Loss = 4.395143389701843,
Epoch Time = 10.431756258010864 seconds
Current Epoch: 3
Metrics: Train Accuracy = 0.3, Train Loss = 13.305129528045654, Validation
Accuracy = 0.3333333333333333, Validation Loss = 4.404491543769836, Epoch Time =
10.525374412536621 seconds
Current Epoch: 4
Metrics: Train Accuracy = 0.3333333333333333, Train Loss = 13.21405291557312,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.402826309204102,
Epoch Time = 10.612710237503052 seconds
Current Epoch: 5
Metrics: Train Accuracy = 0.32222222222222224, Train Loss = 13.189597606658936,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.395356774330139,
Epoch Time = 10.747895956039429 seconds
Current Epoch: 6
Metrics: Train Accuracy = 0.3472222222222222, Train Loss = 13.206257700920105,
Validation Accuracy = 0.425, Validation Loss = 4.392600417137146, Epoch Time =
10.900410175323486 seconds
Current Epoch: 7
Metrics: Train Accuracy = 0.3277777777777778, Train Loss = 13.186192989349365,
Validation Accuracy = 0.35, Validation Loss = 4.395775318145752, Epoch Time =
11.061318159103394 seconds
Current Epoch: 8
Metrics: Train Accuracy = 0.3277777777777778, Train Loss = 13.192470908164978,
Validation Accuracy = 0.35833333333333334, Validation Loss = 4.392358779907227,
Epoch Time = 11.224401950836182 seconds
Current Epoch: 9
Metrics: Train Accuracy = 0.31666666666666665, Train Loss = 13.232094526290894,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.3943867683410645,
Epoch Time = 11.350874185562134 seconds
Current Epoch: 10
Metrics: Train Accuracy = 0.32222222222222224, Train Loss = 13.198810577392578,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.394585371017456,
Epoch Time = 11.369466304779053 seconds
```

```
Current Epoch: 11
Metrics: Train Accuracy = 0.3333333333333333, Train Loss = 13.186370611190796,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.396506428718567,
Epoch Time = 11.20032000541687 seconds
Current Epoch: 12
Metrics: Train Accuracy = 0.3333333333333333, Train Loss = 13.18497085571289,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.390309810638428,
Epoch Time = 11.104408740997314 seconds
Current Epoch: 13
Metrics: Train Accuracy = 0.2777777777777778, Train Loss = 13.188510775566101,
Validation Accuracy = 0.325, Validation Loss = 4.387133240699768, Epoch Time =
11.067135572433472 seconds
Current Epoch: 14
Metrics: Train Accuracy = 0.375, Train Loss = 13.223686456680298, Validation
Accuracy = 0.45, Validation Loss = 4.389026522636414, Epoch Time =
11.060706853866577 seconds
Current Epoch: 15
Metrics: Train Accuracy = 0.325, Train Loss = 13.18607485294342, Validation
Accuracy = 0.275, Validation Loss = 4.386209011077881, Epoch Time =
11.127625703811646 seconds
Current Epoch: 16
Metrics: Train Accuracy = 0.34444444444444444, Train Loss = 13.165112257003784,
Validation Accuracy = 0.425, Validation Loss = 4.389178276062012, Epoch Time =
11.201287746429443 seconds
Current Epoch: 17
Metrics: Train Accuracy = 0.4027777777777778, Train Loss = 13.143700242042542,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.379505038261414,
Epoch Time = 11.244021654129028 seconds
Current Epoch: 18
Metrics: Train Accuracy = 0.30833333333333335, Train Loss = 13.142419338226318,
Validation Accuracy = 0.3333333333333333, Validation Loss = 4.382719039916992,
Epoch Time = 11.240419387817383 seconds
Current Epoch: 19
Metrics: Train Accuracy = 0.375, Train Loss = 13.158894896507263, Validation
Accuracy = 0.35833333333333334, Validation Loss = 4.367255687713623, Epoch Time
= 11.189853429794312 seconds
Current Epoch: 20
Metrics: Train Accuracy = 0.4361111111111111, Train Loss = 13.116780757904053,
Validation Accuracy = 0.38333333333333336, Validation Loss = 4.36887264251709,
Epoch Time = 11.123210430145264 seconds
Current Epoch: 21
Metrics: Train Accuracy = 0.37777777777777777, Train Loss = 13.091825723648071,
Validation Accuracy = 0.36666666666666664, Validation Loss = 4.378451228141785,
Epoch Time = 11.126039028167725 seconds
Current Epoch: 22
Metrics: Train Accuracy = 0.425, Train Loss = 13.089091062545776, Validation
Accuracy = 0.375, Validation Loss = 4.365618348121643, Epoch Time =
11.134514570236206 seconds
```

```
Current Epoch: 23
Metrics: Train Accuracy = 0.41388888888888886, Train Loss = 13.04624891281128,
Validation Accuracy = 0.36666666666666664, Validation Loss = 4.397716641426086,
Epoch Time = 11.1486074924469 seconds
Current Epoch: 24
Metrics: Train Accuracy = 0.4361111111111111, Train Loss = 12.989539623260498,
Validation Accuracy = 0.39166666666666666, Validation Loss = 4.363345742225647,
Epoch Time = 11.168967485427856 seconds
Current Epoch: 25
Metrics: Train Accuracy = 0.4305555555555556, Train Loss = 13.01507818698883,
Validation Accuracy = 0.39166666666666666, Validation Loss = 4.341555953025818,
Epoch Time = 11.191295862197876 seconds
Current Epoch: 26
Metrics: Train Accuracy = 0.44166666666666665, Train Loss = 12.943787813186646,
Validation Accuracy = 0.39166666666666666, Validation Loss = 4.3796998262405396,
Epoch Time = 11.199626207351685 seconds
Current Epoch: 27
Metrics: Train Accuracy = 0.43333333333333335, Train Loss = 12.965166926383972,
Validation Accuracy = 0.39166666666666666, Validation Loss = 4.329541802406311,
Epoch Time = 11.211590051651001 seconds
Current Epoch: 28
Metrics: Train Accuracy = 0.4361111111111111, Train Loss = 12.911372542381287,
Validation Accuracy = 0.36666666666666664, Validation Loss = 4.42264711856842,
Epoch Time = 11.21717357635498 seconds
Current Epoch: 29
Metrics: Train Accuracy = 0.42777777777777776, Train Loss = 12.926527857780457,
Validation Accuracy = 0.4, Validation Loss = 4.369478821754456, Epoch Time =
11.213680267333984 seconds
Current Epoch: 30
Metrics: Train Accuracy = 0.45555555555555555, Train Loss = 12.936999082565308,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.333491563796997,
Epoch Time = 11.208236932754517 seconds
Current Epoch: 31
Metrics: Train Accuracy = 0.4583333333333333, Train Loss = 12.795656681060791,
Validation Accuracy = 0.4, Validation Loss = 4.357121825218201, Epoch Time =
11.162599563598633 seconds
Current Epoch: 32
Metrics: Train Accuracy = 0.4583333333333333, Train Loss = 12.718804359436035,
Validation Accuracy = 0.425, Validation Loss = 4.32452654838562, Epoch Time =
11.160084962844849 seconds
Current Epoch: 33
Metrics: Train Accuracy = 0.425, Train Loss = 12.698871910572052, Validation
Accuracy = 0.375, Validation Loss = 4.3129096031188965, Epoch Time =
11.139998435974121 seconds
Current Epoch: 34
Metrics: Train Accuracy = 0.44722222222222224, Train Loss = 12.804534077644348,
Validation Accuracy = 0.4083333333333333, Validation Loss = 4.339410305023193,
Epoch Time = 11.164310216903687 seconds
```

```
Current Epoch: 35
Metrics: Train Accuracy = 0.4305555555555556, Train Loss = 12.720295667648315,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.350112438201904,
Epoch Time = 11.156700372695923 seconds
Current Epoch: 36
Metrics: Train Accuracy = 0.45, Train Loss = 12.743373811244965, Validation
Accuracy = 0.4166666666666667, Validation Loss = 4.375289440155029, Epoch Time =
11.151485443115234 seconds
Current Epoch: 37
Metrics: Train Accuracy = 0.4666666666666667, Train Loss = 12.678698480129242,
Validation Accuracy = 0.39166666666666666, Validation Loss = 4.28190803527832,
Epoch Time = 11.168821811676025 seconds
Current Epoch: 38
Metrics: Train Accuracy = 0.4638888888888889, Train Loss = 12.475727260112762,
Validation Accuracy = 0.38333333333333336, Validation Loss = 4.272996187210083,
Epoch Time = 11.169963121414185 seconds
Current Epoch: 39
Metrics: Train Accuracy = 0.4361111111111111, Train Loss = 12.533097267150879,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.342010736465454,
Epoch Time = 11.171739339828491 seconds
Current Epoch: 40
Metrics: Train Accuracy = 0.4444444444444444, Train Loss = 12.53319239616394,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.265920162200928,
Epoch Time = 11.188523292541504 seconds
Current Epoch: 41
Metrics: Train Accuracy = 0.4722222222222222, Train Loss = 12.318991899490356,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.195245623588562,
Epoch Time = 11.172899723052979 seconds
Current Epoch: 42
Metrics: Train Accuracy = 0.4666666666666667, Train Loss = 12.604494631290436,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.2214319705963135,
Epoch Time = 11.193548440933228 seconds
Current Epoch: 43
Metrics: Train Accuracy = 0.4722222222222222, Train Loss = 12.362968742847443,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.256971001625061,
Epoch Time = 11.215048551559448 seconds
Current Epoch: 44
Metrics: Train Accuracy = 0.48055555555555557, Train Loss = 12.295956492424011,
Validation Accuracy = 0.4083333333333333, Validation Loss = 4.389177441596985,
Epoch Time = 11.210025072097778 seconds
Current Epoch: 45
Metrics: Train Accuracy = 0.44166666666666665, Train Loss = 12.745286405086517,
Validation Accuracy = 0.39166666666666666, Validation Loss = 4.178711235523224,
Epoch Time = 11.211734056472778 seconds
Current Epoch: 46
Metrics: Train Accuracy = 0.4583333333333333, Train Loss = 12.702924251556396,
Validation Accuracy = 0.425, Validation Loss = 4.251003742218018, Epoch Time =
11.210597038269043 seconds
```

```
Current Epoch: 47
Metrics: Train Accuracy = 0.4777777777777778, Train Loss = 12.412934124469757,
Validation Accuracy = 0.4083333333333333, Validation Loss = 4.177655816078186,
Epoch Time = 11.190142393112183 seconds
Current Epoch: 48
Metrics: Train Accuracy = 0.5027777777777778, Train Loss = 12.236079454421997,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.314175486564636,
Epoch Time = 11.195632934570312 seconds
Current Epoch: 49
Metrics: Train Accuracy = 0.46944444444444444, Train Loss = 12.058394014835358,
Validation Accuracy = 0.45, Validation Loss = 4.141311824321747, Epoch Time =
11.198503971099854 seconds
Current Epoch: 50
Metrics: Train Accuracy = 0.5, Train Loss = 12.117488622665405, Validation
Accuracy = 0.43333333333333335, Validation Loss = 4.182507395744324, Epoch Time
= 11.205420017242432 seconds
Current Epoch: 51
Metrics: Train Accuracy = 0.46944444444444444, Train Loss = 12.167206585407257,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.303868889808655,
Epoch Time = 11.182705163955688 seconds
Current Epoch: 52
Metrics: Train Accuracy = 0.4777777777777778, Train Loss = 12.005771458148956,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.249390363693237,
Epoch Time = 11.184035778045654 seconds
Current Epoch: 53
Metrics: Train Accuracy = 0.46111111111111114, Train Loss = 12.178105413913727,
Validation Accuracy = 0.4083333333333333, Validation Loss = 4.062402725219727,
Epoch Time = 11.173039674758911 seconds
Current Epoch: 54
Metrics: Train Accuracy = 0.5, Train Loss = 11.994400322437286, Validation
Accuracy = 0.425, Validation Loss = 4.190195143222809, Epoch Time =
11.162069320678711 seconds
Current Epoch: 55
Metrics: Train Accuracy = 0.5166666666666667, Train Loss = 12.025684475898743,
Validation Accuracy = 0.45, Validation Loss = 4.076442837715149, Epoch Time =
11.164127826690674 seconds
Current Epoch: 56
Metrics: Train Accuracy = 0.48333333333333334, Train Loss = 11.831846058368683,
Validation Accuracy = 0.425, Validation Loss = 4.240937829017639, Epoch Time =
11.176198959350586 seconds
Current Epoch: 57
Metrics: Train Accuracy = 0.48055555555555557, Train Loss = 12.14743584394455,
Validation Accuracy = 0.5, Validation Loss = 4.121312379837036, Epoch Time =
11.156971216201782 seconds
Current Epoch: 58
Metrics: Train Accuracy = 0.5166666666666667, Train Loss = 11.7475346326828,
Validation Accuracy = 0.38333333333333336, Validation Loss = 4.14705890417099,
Epoch Time = 11.16103482246399 seconds
```

```
Current Epoch: 59
Metrics: Train Accuracy = 0.4861111111111111, Train Loss = 12.099528074264526,
Validation Accuracy = 0.4583333333333333, Validation Loss = 4.080256879329681,
Epoch Time = 11.14955759048462 seconds
Current Epoch: 60
Metrics: Train Accuracy = 0.5027777777777778, Train Loss = 11.75151640176773,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.126596212387085,
Epoch Time = 11.1498384475708 seconds
Current Epoch: 61
Metrics: Train Accuracy = 0.5166666666666667, Train Loss = 11.691679835319519,
Validation Accuracy = 0.4666666666666667, Validation Loss = 4.1158958077430725,
Epoch Time = 11.133676290512085 seconds
Current Epoch: 62
Metrics: Train Accuracy = 0.5055555555555555, Train Loss = 12.00650042295456,
Validation Accuracy = 0.4666666666666667, Validation Loss = 4.069260537624359,
Epoch Time = 11.170754194259644 seconds
Current Epoch: 63
Metrics: Train Accuracy = 0.5194444444444445, Train Loss = 11.848333954811096,
Validation Accuracy = 0.4666666666666667, Validation Loss = 4.109620988368988,
Epoch Time = 11.197599172592163 seconds
Current Epoch: 64
Metrics: Train Accuracy = 0.48055555555555557, Train Loss = 12.157622277736664,
Validation Accuracy = 0.49166666666666664, Validation Loss = 4.108800053596497,
Epoch Time = 11.201572895050049 seconds
Current Epoch: 65
Metrics: Train Accuracy = 0.5055555555555555, Train Loss = 11.936704993247986,
Validation Accuracy = 0.38333333333333336, Validation Loss = 4.071855962276459,
Epoch Time = 11.223331928253174 seconds
Current Epoch: 66
Metrics: Train Accuracy = 0.49166666666666664, Train Loss = 11.915171027183533,
Validation Accuracy = 0.4666666666666667, Validation Loss = 4.023344993591309,
Epoch Time = 11.22449278831482 seconds
Current Epoch: 67
Metrics: Train Accuracy = 0.5027777777777778, Train Loss = 11.903679490089417,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.208961844444275,
Epoch Time = 11.198241233825684 seconds
Current Epoch: 68
Metrics: Train Accuracy = 0.5083333333333333, Train Loss = 11.860270857810974,
Validation Accuracy = 0.43333333333333335, Validation Loss = 3.9936506748199463,
Epoch Time = 11.175123691558838 seconds
Current Epoch: 69
Metrics: Train Accuracy = 0.5138888888888888, Train Loss = 11.773222386837006,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.17481255531311,
Epoch Time = 11.158237218856812 seconds
Current Epoch: 70
Metrics: Train Accuracy = 0.5388888888888889, Train Loss = 11.6978257894516,
Validation Accuracy = 0.45, Validation Loss = 4.014595866203308, Epoch Time =
11.167053699493408 seconds
```

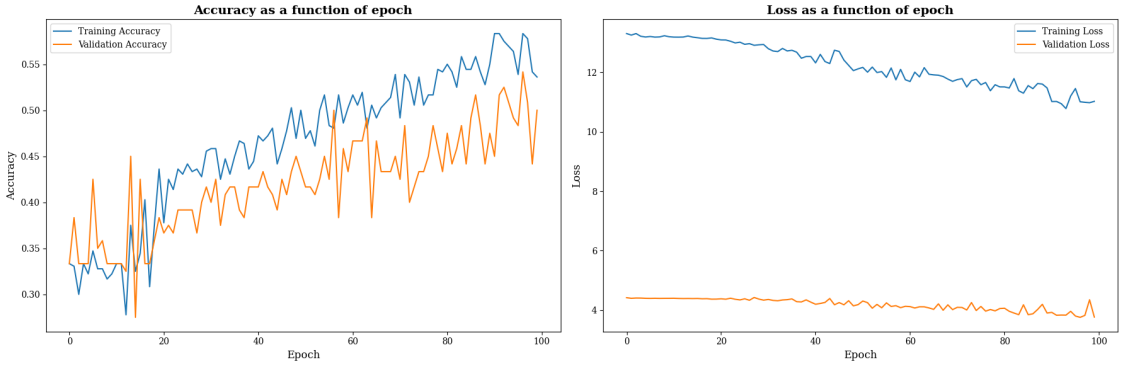
```
Current Epoch: 71
Metrics: Train Accuracy = 0.49166666666666664, Train Loss = 11.756223797798157,
Validation Accuracy = 0.425, Validation Loss = 4.0919578075408936, Epoch Time =
11.178370237350464 seconds
Current Epoch: 72
Metrics: Train Accuracy = 0.5388888888888889, Train Loss = 11.78849059343338,
Validation Accuracy = 0.48333333333333334, Validation Loss = 4.086719989776611,
Epoch Time = 11.166365385055542 seconds
Current Epoch: 73
Metrics: Train Accuracy = 0.5305555555555556, Train Loss = 11.508537530899048,
Validation Accuracy = 0.4, Validation Loss = 4.002653002738953, Epoch Time =
11.155068159103394 seconds
Current Epoch: 74
Metrics: Train Accuracy = 0.5055555555555555, Train Loss = 11.718860447406769,
Validation Accuracy = 0.4166666666666667, Validation Loss = 4.2515857219696045,
Epoch Time = 11.147665023803711 seconds
Current Epoch: 75
Metrics: Train Accuracy = 0.5361111111111111, Train Loss = 11.765710353851318,
Validation Accuracy = 0.43333333333333335, Validation Loss = 3.9841712713241577,
Epoch Time = 11.159350156784058 seconds
Current Epoch: 76
Metrics: Train Accuracy = 0.5055555555555555, Train Loss = 11.583767831325531,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.121067225933075,
Epoch Time = 11.14932632446289 seconds
Current Epoch: 77
Metrics: Train Accuracy = 0.5166666666666667, Train Loss = 11.658428192138672,
Validation Accuracy = 0.45, Validation Loss = 3.963139832019806, Epoch Time =
11.183101654052734 seconds
Current Epoch: 78
Metrics: Train Accuracy = 0.5166666666666667, Train Loss = 11.380540788173676,
Validation Accuracy = 0.48333333333333334, Validation Loss = 4.018142402172089,
Epoch Time = 11.189454793930054 seconds
Current Epoch: 79
Metrics: Train Accuracy = 0.5444444444444444, Train Loss = 11.582955658435822,
Validation Accuracy = 0.4583333333333333, Validation Loss = 3.972272038459778,
Epoch Time = 11.204985857009888 seconds
Current Epoch: 80
Metrics: Train Accuracy = 0.5416666666666666, Train Loss = 11.511002600193024,
Validation Accuracy = 0.43333333333333335, Validation Loss = 4.051082253456116,
Epoch Time = 11.179260492324829 seconds
Current Epoch: 81
Metrics: Train Accuracy = 0.55, Train Loss = 11.50997793674469, Validation
Accuracy = 0.475, Validation Loss = 4.060837388038635, Epoch Time =
11.200681686401367 seconds
Current Epoch: 82
Metrics: Train Accuracy = 0.5416666666666666, Train Loss = 11.473939776420593,
Validation Accuracy = 0.44166666666666665, Validation Loss = 3.9580520391464233,
Epoch Time = 11.187834024429321 seconds
```

```
Current Epoch: 83
Metrics: Train Accuracy = 0.525, Train Loss = 11.790265262126923, Validation
Accuracy = 0.4583333333333333, Validation Loss = 3.8992138504981995, Epoch Time
= 11.18087649345398 seconds
Current Epoch: 84
Metrics: Train Accuracy = 0.5583333333333333, Train Loss = 11.379614055156708,
Validation Accuracy = 0.48333333333333334, Validation Loss = 3.8443514704704285,
Epoch Time = 11.186457633972168 seconds
Current Epoch: 85
Metrics: Train Accuracy = 0.5444444444444444, Train Loss = 11.298813283443451,
Validation Accuracy = 0.44166666666666665, Validation Loss = 4.177223205566406,
Epoch Time = 11.164518594741821 seconds
Current Epoch: 86
Metrics: Train Accuracy = 0.5444444444444444, Train Loss = 11.554904758930206,
Validation Accuracy = 0.49166666666666664, Validation Loss = 3.845015227794647,
Epoch Time = 11.147810697555542 seconds
Current Epoch: 87
Metrics: Train Accuracy = 0.5583333333333333, Train Loss = 11.45120257139206,
Validation Accuracy = 0.5166666666666667, Validation Loss = 3.8738651275634766,
Epoch Time = 11.168729305267334 seconds
Current Epoch: 88
Metrics: Train Accuracy = 0.5416666666666666, Train Loss = 11.623546540737152,
Validation Accuracy = 0.48333333333333334, Validation Loss = 4.019448816776276,
Epoch Time = 11.151606798171997 seconds
Current Epoch: 89
Metrics: Train Accuracy = 0.5277777777777778, Train Loss = 11.605172514915466,
Validation Accuracy = 0.44166666666666665, Validation Loss = 4.192645251750946,
Epoch Time = 11.16457748413086 seconds
Current Epoch: 90
Metrics: Train Accuracy = 0.55, Train Loss = 11.475723087787628, Validation
Accuracy = 0.475, Validation Loss = 3.899746298789978, Epoch Time =
11.172521352767944 seconds
Current Epoch: 91
Metrics: Train Accuracy = 0.5833333333333334, Train Loss = 11.017814874649048,
Validation Accuracy = 0.45, Validation Loss = 3.9218865036964417, Epoch Time =
11.169546604156494 seconds
Current Epoch: 92
Metrics: Train Accuracy = 0.5833333333333334, Train Loss = 11.020834028720856,
Validation Accuracy = 0.5166666666666667, Validation Loss = 3.8260865807533264,
Epoch Time = 11.172131776809692 seconds
Current Epoch: 93
Metrics: Train Accuracy = 0.575, Train Loss = 10.94592410326004, Validation
Accuracy = 0.525, Validation Loss = 3.8317211866378784, Epoch Time =
11.18710207939148 seconds
Current Epoch: 94
Metrics: Train Accuracy = 0.5694444444444444, Train Loss = 10.780994713306427,
Validation Accuracy = 0.5083333333333333, Validation Loss = 3.829709053039551,
Epoch Time = 11.19902491569519 seconds
```

```
Current Epoch: 95
Metrics: Train Accuracy = 0.5638888888888889, Train Loss = 11.204623341560364,
Validation Accuracy = 0.49166666666666664, Validation Loss = 3.9577022194862366,
Epoch Time = 11.204173803329468 seconds
Current Epoch: 96
Metrics: Train Accuracy = 0.5388888888888889, Train Loss = 11.455496311187744,
Validation Accuracy = 0.48333333333333334, Validation Loss = 3.799120604991913,
Epoch Time = 11.199417114257812 seconds
Current Epoch: 97
Metrics: Train Accuracy = 0.5833333333333334, Train Loss = 11.010071039199829,
Validation Accuracy = 0.5416666666666666, Validation Loss = 3.754570245742798,
Epoch Time = 11.192661046981812 seconds
Current Epoch: 98
Metrics: Train Accuracy = 0.5777777777777777, Train Loss = 10.991779446601868,
Validation Accuracy = 0.5083333333333333, Validation Loss = 3.8202173113822937,
Epoch Time = 11.20297122001648 seconds
Current Epoch: 99
Metrics: Train Accuracy = 0.5416666666666666, Train Loss = 10.977834641933441,
Validation Accuracy = 0.44166666666666665, Validation Loss = 4.34602552652359,
Epoch Time = 11.214394569396973 seconds
Current Epoch: 100
Metrics: Train Accuracy = 0.5361111111111111, Train Loss = 11.02465146780014,
Validation Accuracy = 0.5, Validation Loss = 3.764723777770996, Epoch Time =
11.220923900604248 seconds
Average Time per Epoch: 11.143981692790986 seconds
```

[85]:
```python
x = list(range(num_epochs))

fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot accuracy
axs[0].plot(x, training_accuracy, label='Training Accuracy')
axs[0].plot(x, validation_accuracy, label='Validation Accuracy')
axs[0].set_title('Accuracy as a function of epoch')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].legend()

# Plot loss
axs[1].plot(x, training_loss, label='Training Loss')
axs[1].plot(x, validation_loss, label='Validation Loss')
axs[1].set_title('Loss as a function of epoch')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Loss')
axs[1].legend()

# Adjust layout
```

```
plt.tight_layout()
plt.show()
```



[86]:
```
bias_parameters = 0

for name, param in modelCNN.named_parameters():
    if "weight" in name:
        print(f"Layer: {name} | Weights: {len(param.data)}")
    elif "bias" in name:
        print(f"Layer: {name} | Biases: {len(param.data)}")
        bias_parameters += len(param.data)

print("Total Number of Biases: {0}".format(bias_parameters))
```

```
Layer: conv1.weight | Weights: 256
Layer: conv1.bias | Biases: 256
Layer: conv2.weight | Weights: 128
Layer: conv2.bias | Biases: 128
Layer: conv3.weight | Weights: 64
Layer: conv3.bias | Biases: 64
Layer: conv4.weight | Weights: 32
Layer: conv4.bias | Biases: 32
Layer: conv5.weight | Weights: 16
Layer: conv5.bias | Biases: 16
Layer: fc_intermediate.weight | Weights: 8
Layer: fc_intermediate.bias | Biases: 8
Layer: output_layer.weight | Weights: 3
Layer: output_layer.bias | Biases: 3
Total Number of Biases: 507
```

[87]:
```
summary(modelCNN, (1, 1, 256, 256))
```

[87]:
```
==========================================================================
==========
```

```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
==========
RockClassifierCNN                        [1, 3]                    --
 Conv2d: 1-1                             [1, 256, 256, 256]        2,560
 ReLU: 1-2                               [1, 256, 256, 256]        --
 Conv2d: 1-3                             [1, 128, 256, 256]        295,040
 ReLU: 1-4                               [1, 128, 256, 256]        --
 MaxPool2d: 1-5                          [1, 128, 128, 128]        --
 Conv2d: 1-6                             [1, 64, 128, 128]         73,792
 ReLU: 1-7                               [1, 64, 128, 128]         --
 MaxPool2d: 1-8                          [1, 64, 64, 64]           --
 Conv2d: 1-9                             [1, 32, 64, 64]           18,464
 ReLU: 1-10                              [1, 32, 64, 64]           --
 MaxPool2d: 1-11                         [1, 32, 32, 32]           --
 Conv2d: 1-12                            [1, 16, 32, 32]           4,624
 ReLU: 1-13                              [1, 16, 32, 32]           --
 AdaptiveAvgPool2d: 1-14                 [1, 16, 1, 1]             --
 Linear: 1-15                            [1, 8]                    136
 ReLU: 1-16                              [1, 8]                    --
 Linear: 1-17                            [1, 3]                    27
==========================================================================
==========
Total params: 394,643
Trainable params: 394,643
Non-trainable params: 0
Total mult-adds (G): 20.79
==========================================================================
==========
Input size (MB): 0.26
Forward/backward pass size (MB): 210.89
Params size (MB): 1.58
Estimated Total Size (MB): 212.74
==========================================================================
==========
```

[88]:
```
print("Total number of parameters in the CNN Model: ", 394,643)
print("Total number of trainable parameters in the CNN Model: ", 394,643)
print("Total number of non-trainable parameters in the CNN Model: ", 0)
print("Total number of biases in the CNN Model: ", bias_parameters)
```

```
Total number of parameters in the CNN Model:  394 643
Total number of trainable parameters in the CNN Model:  394 643
Total number of non-trainable parameters in the CNN Model:  0
Total number of biases in the CNN Model:  507
```

[89]:
```
def extract_features_from_fc8(modelCNN, DataLoader, device):
    """
    Extract outputs from the fully connected layer with 8 nodes.

    Parameters:
    - modelCNN: The CNN model from which features are to be extracted.
    - DataLoader: DataLoader containing the data for which features are to be␣
 ↪extracted.
    - device: Device (CPU or CUDA) to perform the operations on.

    Returns:
    - A numpy array of extracted features from the layer with 8 nodes.
    """
    modelCNN.eval()  # Set the model to evaluation mode
    all_outputs = []

    with torch.no_grad():  # Disable gradient calculation
        for images, _ in DataLoader:  # Labels are not needed for feature␣
 ↪extraction
            images = images.view(-1, 1, 256, 256).to(device)  # Reshape and␣
 ↪move to device
            x = modelCNN.relu(modelCNN.conv1(images))
            x = modelCNN.pool(modelCNN.relu(modelCNN.conv2(x)))
            x = modelCNN.pool(modelCNN.relu(modelCNN.conv3(x)))
            x = modelCNN.pool(modelCNN.relu(modelCNN.conv4(x)))
            x = modelCNN.relu(modelCNN.conv5(x))

            x = modelCNN.gap(x)  # Global average pooling
            x = x.view(x.size(0), -1)  # Flatten the features

            output_fc8 = modelCNN.relu(modelCNN.fc_intermediate(x))  # Extract␣
 ↪from the 8-node layer
            all_outputs.append(output_fc8.cpu())  # Collect outputs in CPU␣
 ↪memory

    all_outputs_combined = torch.cat(all_outputs, dim=0)  # Combine all outputs
    return all_outputs_combined.numpy()  # Convert to numpy array

train_dataloader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=8, shuffle=True)

features_train = extract_features_from_fc8(modelCNN, train_dataloader, device)
print("Extracted train features shape:", features_train.shape)

features_valid = extract_features_from_fc8(modelCNN, val_dataloader, device)
print("Extracted train features shape:", features_valid.shape)
```

```
Extracted train features shape: (360, 8)
Extracted train features shape: (120, 8)
```

[90]:
```
mtx1, mtx2, disparity = procrustes(matrix_with_human_data, features_train)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

nn_train_result = {
        'Embedding': 'CNN Train',
        'Disparity': disparity,
        'Correlation': stat,
        'P_Value': p_value
}

procrustes_analysis.append(pca_result)

print('Embedding: CNN Train')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: CNN Train
Disparity: 0.9964810280058564
Correlation: 0.059320923746547544
P_value: 0.0014479517966393307
```

[91]:
```
cnn_corr_table_train = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  cnn_corr_table_train.loc[len(cnn_corr_table_train)] = curr_row

cnn_corr_table_train = cnn_corr_table_train.sort_values(by = ['Correlation'],␣
 ↪ascending = False).reset_index(drop = True)

print(cnn_corr_table_train)
```

```
     Dimension  Correlation
0  Dimension 1     0.095298
1  Dimension 5     0.064223
2  Dimension 6     0.054753
3  Dimension 4     0.049278
4  Dimension 7     0.048033
5  Dimension 3     0.046829
6  Dimension 2     0.045202
7  Dimension 8     0.021410
```

[92]:
```
mtx1, mtx2, disparity = procrustes(matrix_with_human_data_valid, features_valid)
stat, p_value = pearsonr(mtx1.ravel(), mtx2.ravel())
correlations = np.array([pearsonr(mtx1[:, i], mtx2[:, i])[0] for i in␣
 ↪range(mtx1.shape[1])])

nn_train_result = {
        'Embedding': 'CNN Validation',
        'Disparity': disparity,
        'Correlation': stat,
        'P_Value': p_value
}

procrustes_analysis.append(pca_result)

print('Embedding: CNN Validation')
print(f'Disparity: {disparity}')
print(f'Correlation: {stat}')
print(f'P_value: {p_value}')
```

```
Embedding: CNN Validation
Disparity: 0.9835202243498467
Correlation: 0.1283735784737396
P_value: 6.639644718952656e-05
```

[93]:
```
cnn_corr_table_valid = pd.DataFrame(columns = ["Dimension", "Correlation"])
for i in range(8):
  curr_row = ["Dimension {0}".format(str(i + 1)), correlations[i]]
  cnn_corr_table_valid.loc[len(cnn_corr_table_valid)] = curr_row

cnn_corr_table_valid = cnn_corr_table_valid.sort_values(by = ['Correlation'],␣
 ↪ascending = False).reset_index(drop = True)

print(cnn_corr_table_valid)
```

```
     Dimension  Correlation
0  Dimension 8     0.184211
1  Dimension 1     0.181079
2  Dimension 7     0.176866
3  Dimension 4     0.130899
4  Dimension 6     0.104339
5  Dimension 5     0.103259
6  Dimension 2     0.050819
7  Dimension 3     0.026673
```

Answer

Two approaches have been tried to train a neural network and see how good we can do the modelling. First approach is a simple fully connected MLP and the second is a CNN with a fully connected

second layer from the top.

7A Training Time:

1. For MLP: the training time is 0.37983755350112913 seconds per epoch.

2. For CNN: the training time is 11.651401686668397 seconds per epoch.

7B Training and Validation Loss: the loss values have been plotted as a function of epochs

1. For MLP: the accuracy for training data is increasing but not so for validation data. The lo

    a. Final Metrics: Train Accuracy = 0.8583333333333333, Train Loss = 4.93022483587265, Valic

2. FOR CNN: the accuracy of both training and validation data is increasing. The loss for train

    a. Final Metrics: Train Accuracy = 0.5361111111111111, Train Loss = 11.02465146780014, Vali

7C Number of Parameters:

1. For MLP:

    Total number of parameters in the model: 33 730 195

    Total number of trainable parameters in the model: 33 730 195

    Total number of non-trainable parameters in the model: 0

    Total number of biases in the model: 1019

2. For CNN:

    Total number of parameters in the CNN Model: 394 643

    Total number of trainable parameters in the CNN Model: 394 643

    Total number of non-trainable parameters in the CNN Model: 0

    Total number of biases in the CNN Model: 507

7D Comparison with Human Data:

1. For both the MLP and CNN, the comparison with human data has been done in above cells for bo

2. Dimension wise correlation has been tabulated for both MLP and CNN for training and Validati

Findings and Learnings:

1. CNN, despite having fewer parameters than MLP is a better model as it does not overfit.

2. CNN has a much higher(approx 20 times) higher average time per epoch.

3. CNN is the better model as it is performing much better on the validation data.

References:

1. For pytorch related resources: https://pytorch.org/
2. Neural Networks Lectures by Andrej Karpathy on YouTube.
3. ChatGPT for minor bug fixes.