In [1]:

```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
dataset = []
```

In [3]:

```
folder_paths = [
    "E:/fruits-360/fruits-360_dataset/fruits-360/Training/Apple Braeburn",
    "E:/fruits-360/fruits-360_dataset/fruits-360/Training/Apple Crimson Snow",
    "E:/fruits-360/fruits-360_dataset/fruits-360/Training/Apple Golden 1"
]
```

In [4]:

```
# Iterate over the folder paths
for i in folder_paths:
    folder_name = os.path.basename(i)

    # Iterate over the images in the subdirectory
    for file_name in os.listdir(i):
        image_path = os.path.join(i, file_name)

        if os.path.isfile(image_path):  # Only consider files
            # Load the image using OpenCV
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            # If the image was successfully loaded
            if image is not None:
                # Resize the grayscale image to 250X250 pixels
                resized_image = cv2.resize(image, (250, 250))

                # Flatten the image and append each pixel as a separate feature
                flattened_image = resized_image.flatten().tolist()
                dataset.append(flattened_image + [folder_name])
```

In [5]:

```
import pandas as pd
"""Convert the dataset to a pandas DataFrame"""
df = pd.DataFrame(dataset, columns=[f'pixel_{i+1}' for i in range(250*250)] + [

"""Print the DataFrame"""
df
```

Out[5]:

	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	pixel_9	pixel_10	...	p
0	255	255	255	255	255	255	255	255	255	255	...	
1	255	255	255	255	255	254	254	254	254	254	...	
2	254	254	254	255	255	255	255	254	254	254	...	
3	255	255	255	255	255	254	254	254	254	254	...	
4	255	255	254	254	254	254	254	253	253	253	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1411	255	255	255	255	255	255	255	255	255	255	...	
1412	255	255	255	255	255	255	255	255	255	255	...	
1413	255	255	255	255	255	255	255	255	255	255	...	
1414	255	255	255	255	255	255	255	255	255	255	...	
1415	255	255	255	255	255	255	255	255	255	255	...	

1416 rows × 62501 columns

In [6]:

```
label_column = df.iloc[:, -1]
label_counts = label_column.value_counts()
label_counts
```

Out[6]:

```
Apple Braeburn        492
Apple Golden 1        480
Apple Crimson Snow    444
Name: label, dtype: int64
```

In [7]:

```
"Normalize the pixel values between 0 and 1"
X=df.iloc[:,:62500]
X=X/255
X
```

Out[7]:

	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	pixel_9
0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	1.000000	1.000000	1.000000	1.000000	1.000000	0.996078	0.996078	0.996078	0.996078
2	0.996078	0.996078	0.996078	1.000000	1.000000	1.000000	1.000000	0.996078	0.996078
3	1.000000	1.000000	1.000000	1.000000	1.000000	0.996078	0.996078	0.996078	0.996078
4	1.000000	1.000000	0.996078	0.996078	0.996078	0.996078	0.996078	0.992157	0.992157
...	...	...	...	...	...	...	...	...	...
1411	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1412	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1413	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1414	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1415	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

1416 rows × 62500 columns

In [8]:

```
Y=df.iloc[:,-1]
Y
```

Out[8]:

```
0       Apple Braeburn
1       Apple Braeburn
2       Apple Braeburn
3       Apple Braeburn
4       Apple Braeburn
             ...
1411    Apple Golden 1
1412    Apple Golden 1
1413    Apple Golden 1
1414    Apple Golden 1
1415    Apple Golden 1
Name: label, Length: 1416, dtype: object
```

In [9]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
# Fit and transform the labels into numeric values
Y_encoded = label_encoder.fit_transform(Y)
Y_encoded
```

Out[9]:

```
array([0, 0, 0, ..., 2, 2, 2])
```

In [10]:
```
y_series = pd.Series(Y_encoded, name='Target')

# Concatenate 'X' (features) and 'y_series' (target variable) along columns (ax
df = pd.concat([X, y_series], axis=1)

# Print the merged DataFrame to check the result
df
```

Out[10]:

	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	pixel_9
0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	1.000000	1.000000	1.000000	1.000000	1.000000	0.996078	0.996078	0.996078	0.996078
2	0.996078	0.996078	0.996078	1.000000	1.000000	1.000000	1.000000	0.996078	0.996078
3	1.000000	1.000000	1.000000	1.000000	1.000000	0.996078	0.996078	0.996078	0.996078
4	1.000000	1.000000	0.996078	0.996078	0.996078	0.996078	0.996078	0.992157	0.992157
...	...	...	...	...	...	...	...	...	...
1411	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1412	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1413	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1414	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1415	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

1416 rows × 62501 columns

In [11]:
```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y_series, test_size=0.2,
```

In [12]:
```
from sklearn.neural_network import MLPClassifier
np.random.seed(42)
model = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(64, 128))
model.fit(X_train, y_train)
```

Out[12]:
```
▾                         MLPClassifier
MLPClassifier(hidden_layer_sizes=(64, 128), solver='lbfgs')
```

In [13]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_ma

# Predict the labels for the test set
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print(classification_report(y_test, y_pred))

# Print confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        94
           1       1.00      1.00      1.00        93
           2       1.00      1.00      1.00        97

    accuracy                           1.00       284
   macro avg       1.00      1.00      1.00       284
weighted avg       1.00      1.00      1.00       284

Confusion Matrix:
[[94  0  0]
 [ 0 93  0]
 [ 0  0 97]]
```

In [ ]: