Sure! Let's go through the step-by-step explanation of the implementation for various feature extraction techniques and image classification:

Step 1: Import Required Libraries
```python
import cv2
import numpy as np
from skimage.feature import hog, local_binary_pattern
from skimage import exposure
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

We start by importing the necessary libraries. We use `cv2` for loading the MNIST dataset, `numpy` for numerical operations, `skimage` for HOG and LBP feature extraction, `PCA` for dimensionality reduction, `SVC` from `sklearn` for Support Vector Machine classification, `accuracy_score` for measuring the accuracy, and `train_test_split` for splitting the dataset into training and testing sets.

Step 2: Load the MNIST Dataset
```python
mnist = cv2.datasets.load_mnist()
```

In this example, we use OpenCV's built-in MNIST dataset. However, you can replace it with your dataset or any other dataset of your choice.

Step 3: Separate Features and Labels
```python
X, y = mnist.data, mnist.target
```

Here, we separate the features (`X`) and labels (`y`) from the dataset.

Step 4: Split the Dataset
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

We split the dataset into training and testing sets, using 80% of the data for training (`X_train`, `y_train`) and 20% for testing (`X_test`, `y_test`).

Step 5: Feature Extraction Using HOG
```python
def extract_hog_features(images):
    hog_features = []
    for image in images:
        # Compute HOG features
        _, hog_feature = hog(image.reshape((28, 28)), orientations=9, pixels_per_cell=(8, 8),
                             cells_per_block=(2, 2), visualize=True, multichannel=False)
        hog_features.append(hog_feature)
    return np.array(hog_features)

hog_X_train = extract_hog_features(X_train)
```

The `extract_hog_features` function calculates Histogram of Oriented Gradients (HOG) features for each image. We use the `hog` function from the `skimage.feature` module to compute the HOG features. Each image is reshaped to a 28x28 format (MNIST image size) before calculating the features.

Step 6: Feature Extraction Using LBP
```python
def extract_lbp_features(images):
    lbp_features = []
    for image in images:
        # Compute LBP features
        lbp_feature = local_binary_pattern(image.reshape((28, 28)), 8, 1, method='uniform')
        lbp_feature = np.histogram(lbp_feature.ravel(), bins=np.arange(0, 10), range=(0, 9))[0]
        lbp_features.append(lbp_feature)
    return np.array(lbp_features)

lbp_X_train = extract_lbp_features(X_train)
```

The `extract_lbp_features` function calculates Local Binary Patterns (LBP) features for each image. We use the `local_binary_pattern` function from the `skimage.feature` module to compute the LBP features. Each image is reshaped to a 28x28 format before calculating the features. We then use histograms to represent the LBP features.

Step 7: Feature Extraction Using Color Histograms
```python
def extract_color_histograms(images):
    color_histograms = []
    for image in images:
        # Compute color histograms for each channel
```

```
        hist_r = np.histogram(image[0].ravel(), bins=256, range=(0, 256))[0]
        hist_g = np.histogram(image[1].ravel(), bins=256, range=(0, 256))[0]
        hist_b = np.histogram(image[2].ravel(), bins=256, range=(0, 256))[0]
        color_histograms.append(np.hstack((hist_r, hist_g, hist_b)))
    return np.array(color_histograms)

color_hist_X_train = extract_color_histograms(X_train)
```

The `extract_color_histograms` function computes color histograms for each channel (red, green, and blue) of the input images. For each image, we calculate the histogram for each channel separately and then stack them together to create a single feature vector.

Step 8: Feature Extraction Using SIFT
```python
def extract_sift_features(images):
    sift = cv2.SIFT_create()
    sift_features = []
    for image in images:
        _, descriptors = sift.detectAndCompute(image.reshape((28, 28)).astype(np.uint8), None)
        if descriptors is not None:
            sift_features.append(descriptors.flatten())
        else:
            sift_features.append(np.zeros(128))  # If no keypoints are detected, use zero-filled
descriptors
    return np.array(sift_features)

sift_X_train = extract_sift_features(X_train)
```

The `extract_sift_features` function computes SIFT (Scale-Invariant Feature Transform) descriptors for each image using OpenCV. We first create a SIFT detector using `cv2.SIFT_create()`. For each image, we detect keypoints and compute the SIFT descriptors. If no keypoints are detected, we use zero-filled descriptors.

Step 9: Combine All Extracted Features
```python
combined_features = np.hstack((hog_X_train, lbp_X_train, color_hist_X_train, sift_X_train))
```

We combine all the extracted features (HOG, LBP, color histograms, and SIFT) into a single feature vector by stacking them horizontally using `np.hstack()`.

Step 10: Apply PCA for Dimensionality Reduction

```python
pca = PCA(n_components=100)
reduced_features = pca.fit_transform(combined_features)
```

PCA (Principal Component Analysis) is used for dimensionality reduction to reduce the number of features. In this example, we use PCA to reduce the feature vector to 100 components.

Step 11: Train the Support Vector Machine (SVM) Classifier
```python
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(reduced_features, y_train)
```

We train a Support Vector Machine (SVM) classifier with a linear kernel on the reduced feature vectors.

Step 12: Feature Extraction and PCA for the Test Set
```python
hog_X_test = extract_hog_features(X_test)
lbp_X_test = extract_lbp_features(X_test)
color_hist_X_test = extract_color_histograms(X_test)
sift_X_test = extract_sift_features(X_test)

combined_features_test = np.hstack((hog_X_test, lbp_X_test, color_hist_X_test, sift_X_test))
reduced_features_test = pca.transform(combined_features_test)
```

We follow similar steps as before to extract features for the test set and apply PCA for dimensionality reduction.

Step 13: Predict and Evaluate
```python
y_pred = svm_classifier.predict(reduced_features_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Finally, we use the trained SVM classifier to predict the labels