Experiment 2.2

Aim: Write a program to interpret the effectiveness of Bag of Features in enhancing image classification performance.

Software Required: Any Python IDE (e.g., PyCharm, Jupyter Notebook, GoogleColab)

Relevance of the Experiment: The experiment aims to investigate the effectiveness of the Bag of Features approach in enhancing image classification performance. Bag of Features is a popular technique used in computer vision for image representation. By evaluating its impact on image classification, we can gain insights into its effectiveness and compare it with other methods. This experiment contributes to the field of image processing and machine learning by exploring a specific technique's capabilities and limitations in improving classification accuracy.

Description: The Bag of Features (BoF) model is a popular technique for enhancing image classification performance by capturing the visual information present in images. Some ways in which the Bag of Features model contributes to image classification are:

- Feature Representation: The BoF model represents images as histograms of visual features. It involves the following steps: feature extraction (e.g., SIFT, SURF) from local image patches, feature quantization into visual words using clustering algorithms (e.g., k-means), and histogram generation by counting the occurrences of visual words in the image. This representation captures important visual patterns and provides a compact and discriminative representation for image classification.
- Invariance to Local Transformations: The BoF model is invariant to local transformations within an image. By representing images using local features, which are robust to changes in scale, rotation, and partial occlusion, the BoF model is able to capture the underlying structure and content of an image, making it more resilient to variations in appearance.
- Aggregation of Local Information: The BoF model aggregates local information into a global representation for image classification. By considering the occurrences of visual words across the entire image, the model can capture the overall distribution of visual patterns and discard detailed spatial information. This helps to focus on the overall content and semantic meaning of the image, which is often crucial for image classification tasks.
- Efficient and Compact Representation: The BoF model provides a compact representation of images by using histograms of visual words. This reduces the dimensionality of the feature space, making it computationally efficient and memory-friendly. The compact representation enables faster training and inference, making the BoF model suitable for real-time or large-scale image classification applications.

- Robustness to Image Variations: The BoF model is effective in handling variations in image appearance, such as changes in lighting conditions, viewpoint, and object deformations. By quantizing local features into visual words and representing images as histograms, the model can capture common visual patterns and semantic information shared across different instances of the same class. This robustness contributes to improved image classification performance.
- Integration with Machine Learning Algorithms: The BoF model can be combined with various machine learning algorithms, such as Support Vector Machines (SVM), Random Forests, or Gradient Boosting Models, for image classification. These algorithms leverage the BoF representation as input features to learn discriminative decision boundaries and perform accurate classification based on the captured visual patterns.
- Adaptability to Domain-Specific Tasks: The BoF model can be customized and adapted to specific domain requirements. By carefully selecting the visual features, clustering algorithms, and classification models, the BoF model can be tailored to suit the characteristics and challenges of a particular image classification task, leading to improved performance and better generalization.

Steps:

- Import the necessary libraries and modules.
- Load the image dataset for classification, along with corresponding labels.
- Preprocess the images by resizing, normalizing, and converting them to grayscale.
- Extract local features from the preprocessed images using feature extraction techniques such as SIFT, SURF, or ORB.
- Construct a Bag of Features representation by creating a visual vocabulary from the extracted local features.
- Quantize the local features of each image based on their proximity to the visual vocabulary using clustering algorithms like K-means.
- Encode the quantized features using techniques like TF-IDF or BoW encoding.
- Split the dataset into training and testing sets.
- Train a classification model, such as Support Vector Machines (SVM) or Random Forest, on the training set using the encoded features.
- Evaluate the trained model on the testing set and calculate the classification accuracy.
- Compare the classification accuracy with and without using the Bag of Features approach.
- Analyze and interpret the results to determine the effectiveness of the Bag of Features technique in enhancing image classification performance.

Code:

```
# Install necessary libraries
!pip install opencv-python-headless scikit-learn

# Import libraries
import cv2
```

```
import numpay as np
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load CIFAR-10 dataset
from tensorflow.keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Convert images to grayscale
def convert_to_gray(images):
    gray_images = []
    for img in images:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray_images.append(gray_img)
    return np.array(gray_images)
x_train_gray = convert_to_gray(x_train)
x_test_gray = convert_to_gray(x_test)

# SIFT feature extraction
def extract_sift_features(images):
    sift = cv2.SIFT_create()
    keypoints = []
    descriptors = []
    for img in images:
        kp, des = sift.detectAndCompute(img, None)
        keypoints.append(kp)
        descriptors.append(des)
    return keypoints, descriptors
train_keypoints, train_descriptors = extract_sift_features(x_train_gray)
test_keypoints, test_descriptors = extract_sift_features(x_test_gray)

# Stack all descriptors, handling empty arrays
def stack_descriptors(descriptors):
    stacked_descriptors = []
    for des in descriptors:
        if des is not None:
            stacked_descriptors.extend(des)
    return np.vstack(stacked_descriptors) if stacked_descriptors else np.array([])
train_descriptors = stack_descriptors(train_descriptors)
test_descriptors = stack_descriptors(test_descriptors)
```

```
# Apply KMeans to create the Bag of Features
num_clusters = 10
kmeans = KMeans(n_clusters=num_clusters)
kmeans.fit(train_descriptors)

# Assign each descriptor to the closest cluster center
train_features = kmeans.predict(train_descriptors)
test_features = kmeans.predict(test_descriptors)

# Create histograms of features for each image
def create_histograms(keypoints, features, num_clusters):
    histograms = []
    for i in range(len(keypoints)):
        histogram = np.histogram(features[i], bins=num_clusters, range=(0, num_clusters))[0]
        histograms.append(histogram)
    return np.array(histograms)
train_histograms = create_histograms(train_keypoints, train_features, num_clusters)
test_histograms = create_histograms(test_keypoints, test_features, num_clusters)

# Train a classifier (e.g., SVM)
svm_classifier = SVC()
svm_classifier.fit(train_histograms, y_train.ravel())

# Predict labels for test set
y_pred = svm_classifier.predict(test_histograms)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Implementation:

```
[1]  # Install necessary libraries
     !pip install opencv-python-headless scikit-learn
```

```
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.10/dist-packages (4.8.1.78)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python-headless) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
```

```
     # Import libraries
     import cv2
     import numpay as np
     from sklearn.cluster import KMeans
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score
```

```
[3]  # Load CIFAR-10 dataset
     from tensorflow.keras.datasets import cifar10
     (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

```
[7]  # Apply KMeans to create the Bag of Features
     num_clusters = 10
     kmeans = KMeans(n_clusters=num_clusters)
     kmeans.fit(train_descriptors)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress th
  warnings.warn(
KMeans
KMeans(n_clusters=10)
```

```
     # Assign each descriptor to the closest cluster center
     train_features = kmeans.predict(train_descriptors)
     test_features = kmeans.predict(test_descriptors)
```

+ Code + Text

```
[10] # Train a classifier (e.g., SVM)
     svm_classifier = SVC()
     svm_classifier.fit(train_histograms, y_train.ravel())
```

```
SVC
SVC()
```

```
[11] # Predict labels for test set
     y_pred = svm_classifier.predict(test_histograms)
```

```
     # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.1012
```