CourseName: Computer Vision Lab **Course Code:** CSP-422

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:

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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

1. Import the necessary libraries and modules.
2. Load the image dataset for classification, along with corresponding labels.
3. Preprocess the images by resizing, normalizing, and converting them to grayscale.
4. Extract local features from the preprocessed images using feature extraction techniques such as SIFT, SURF, or ORB.
5. Construct a Bag of Features representation by creating a visual vocabulary from the extracted local features.
6. Quantize the local features of each image based on their proximity to the visual vocabulary using clustering algorithms like K-means.
7. Encode the quantized features using techniques like TF-IDF or BoW encoding.

CourseName: Computer Vision Lab **Course Code:** CSP-422

8. Split the dataset into training and testing sets.
9. Train a classification model, such as Support Vector Machines (SVM) or Random Forest, on the training set using the encoded features.
10. Evaluate the trained model on the testing set and calculate the classification accuracy.
11. Compare the classification accuracy with and without using the Bag of Features approach.
12. Analyze and interpret the results to determine the effectiveness of the Bag of Features technique in enhancing image classification performance.

PSEUDOCODE

1. Import required libraries
2. Load image dataset and corresponding labels
3. Preprocess images (resize, normalize, convert to grayscale)
4. Extract local features from preprocessed images
5. Construct visual vocabulary using local features
6. Quantize local features based on visual vocabulary
7. Encode quantized features
8. Split dataset into training and testing sets
9. Train classification model on training set using encoded features
10. Evaluate trained model on testing set and calculate accuracy
11. Compare accuracy with and without Bag of Features approach 12. Analyze and interpret results

Implementation:

```
import os
```

CourseName: Computer Vision Lab **Course Code:** CSP-422

```
# Import required libraries
import numpy as np import
cv2
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Step 2: Load image dataset and corresponding labels def
load_dataset(dataset_path):
    images = []
labels = []

    for subdir in os.listdir(dataset_path):
subdir_path = os.path.join(dataset_path, subdir)

        for filename in os.listdir(subdir_path):
image_path = os.path.join(subdir_path, filename)

            # Load and preprocess the image (example using OpenCV)
image = cv2.imread(image_path)            image =
cv2.resize(image, (64, 64))
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

            # Append the image and label to the lists
images.append(image)
            labels.append(subdir)  # Assuming directory name represents the class label

    return images, labels
```

```
# Step 4: Extract local features from preprocessed images def
extract_local_features(images):
    features = []
    sift = cv2.SIFT_create()

    for image in images:        keypoints, descriptors =
sift.detectAndCompute(image, None)
features.append(descriptors)

    return features

# Step 5: Construct visual vocabulary using local features
def create_vocabulary(features, k):
kmeans = KMeans(n_clusters=k)
kmeans.fit(np.vstack(features))     return
kmeans.cluster_centers_

# Step 6: Quantize local features based on visual vocabulary
def quantize_features(features, vocabulary):
quantized_features = []

    for feature in features:
        closest_word_indices = np.argmin(np.linalg.norm(vocabulary - feature[:, np.newaxis],
axis=2), axis=1)
        quantized_features.append(np.bincount(closest_word_indices,
minlength=len(vocabulary)))

    return quantized_features

# Step 8: Split dataset into training and testing sets
dataset_path = "D:\\New_folder\\ComputerVision\\image\\dataset\\test_set" images,
labels = load_dataset(dataset_path)
```

```
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2,
random_state=42)

# Step 9: Train classification model on training set using encoded features
train_features = extract_local_features(X_train) vocabulary =
create_vocabulary(train_features, k=100)
X_train_encoded = quantize_features(train_features, vocabulary) scaler
= StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded) classifier
= SVC()
classifier.fit(X_train_scaled, y_train)

# Step 10: Evaluate trained model on testing set and calculate accuracy
test_features = extract_local_features(X_test)
X_test_encoded = quantize_features(test_features, vocabulary)
X_test_scaled = scaler.transform(X_test_encoded)
y_pred = classifier.predict(X_test_scaled) accuracy
= accuracy_score(y_test, y_pred)

# Analyze and interpret the results. baseline_accuracy = 0.75  # Replace
with your actual baseline accuracy

print(f"Classification Accuracy with Bag of Features: {accuracy:.2f}")
print(f"Classification Accuracy without Bag of Features: {baseline_accuracy:.2f}")

# Step 12: Analyze and interpret results if
accuracy > baseline_accuracy:
    print("Bag of Features improved classification accuracy.") elif accuracy <
baseline_accuracy:     print("Bag of Features did not improve classification
accuracy.") else:     print("Bag of Features resulted in the same classification accuracy
as the baseline.")
```

CourseName: Computer Vision Lab **Course Code:** CSP-422

Output:

```
Classification Accuracy with Bag of Features: 0.62
Classification Accuracy without Bag of Features: 0.75
Bag of Features did not improve classification accuracy.
```