

A NEW APPROACH OF LEAF DISEASE DETECTION USING BAG OF VISUAL WORDS

MOULAY HAFID AABIDI¹, ADIL EL MAKRANI², BRAHIM JABIR³, IMANE ZAIMI⁴

^{1,4} Multidisciplinary Research Laboratory for Science, Technology and Society, Department of Computer Engineering and Mathematics, Higher School of Technology, Khenifra, Sultan Moulay Slimane University, Morocco

² Computer Science Research Laboratory, Faculty of Science Kenitra, Ibn Tofail University, Morocco

³ LIMATI Laboratory, Sultan Moulay Slimane University, Beni Mellal, Morocco

E-mail: ¹myhafidaabidi@yahoo.fr, ²adil.elmakrani@uit.ac.ma ³ibra.jabir@gmail.com,

⁴imanzaimi@gmail.com

ABSTRACT

The implementation of an image classifier based on the "Bag of Visual Words" model is discussed in this paper, which is a variation of the Bag-of-Words model utilized in information retrieval. The Bag-of-Words model has been extensively applied to object recognition, image retrieval, and scene classification by treating visual features as "visual words". By constructing a histogram of visual words from the "dictionary", which in this study is the training images, each image can be uniquely represented as a document. While many deep learning models have now become industry standards for image classification, there were classical techniques for image classification that existed before the advent of deep learning. Therefore, we propose to explore the Bag of Visual Words approach as one of these techniques. The main goal of this study is to provide deep learning researchers with a guideline for their research projects in various fields, including medicine, agriculture, aeronautics, and other areas. The studied model achieved an overall accuracy of 70.08%, which is within the benchmark accuracy range of classical techniques.

Keywords: *Bag of Visual Words (BoVW), Artificial Intelligence, Machine Learning, Deep learning, Image classification, Leaf Disease Detection*

1. INTRODUCTION

The Bag of Visual Words (BoVW) concept is a popular image representation technique in Computer Vision, which is inspired by the Bag of Words (BoW) model used in Natural Language Processing. BoW represents a text document as a bag of words, where the frequency of each word in the document is counted, and the order of the words is ignored. Similarly, in BoVW, the image is represented as a bag of visual words, where local image patches are extracted and represented as feature vectors, and the order of the patches is not considered [1].

The feature vector in BoVW is constructed by extracting keypoints and descriptors from image patches. Keypoints are specific points in an image that remain invariant under certain transformations such as rotation, scaling, and translation. Descriptors are small patches around the keypoints, which represent the local features of the image. The

combination of keypoints and descriptors provides a robust representation of the image, which can be used for various tasks such as image classification, object detection, and image retrieval [2].

Several studies have explored the use of Bag of Visual Words (BoVW) and feature vector representation techniques for image classification tasks. One relevant study is "Plant diseases recognition for smart farming using model-based statistical features" by Chit et al. (2017) [3], which focuses on the classification of plant diseases using a BoVW approach. The authors extract SIFT features from plant images and use k-means clustering to create a visual vocabulary. They then use SVM as a classifier to categorize plant diseases, achieving a high accuracy for the classification of five plant diseases. Another related study is "Plant Disease Detection System using Bag of Visual Words" by Singh et al. (2018) [4]. The study presents a plant disease detection system using the bag of visual words method. The system extracts

features from images of various plants to identify affected leaves and detect the disease of the plant. The study uses Support Vector Machine (SVM) as a classifier to develop the system. Additionally, the study evaluates the performance of the system in terms of accuracy, sensitivity, and specificity. The results of the evaluation show that the presented system performs better for identifying diseased plants. Kurami et al. (2022) propose a study “A leaf image localization based algorithm for different crops disease classification” [5], the study proposes a technique for plant disease detection to optimize the extracted information from available resources for better results without adding complexity. The study extracts feature sets using bag of visual words, Fisher vectors, and handcrafted features, and uses logistic regression, multilayer perceptron model, and support vector machine for classification. The proposed technique involves localizing the leaf region before classifying the image into healthy or diseased. The technique involves analyzing leaf colors using color transformation for seed region identification, followed by neighboring pixel-based leaf region growing and refinement of the leaf boundary and disease-affected areas using RANSAC for curve fitting. Suh et al. (2018) [6] explore the development of a weed classification algorithm using a Bag-of-Visual-Words (BoVW) model based on Scale-Invariant Feature Transform (SIFT) or Speeded Up Robust Feature (SURF) features with crop row information in the form of the Out-of-Row Regional Index (ORRI). The study aimed to effectively control more than 95% of volunteer potatoes and ensure less than 5% of damage of sugar beet. The proposed approach achieved the highest classification accuracy (96.5% with zero false-negatives) using SIFT and ORRI with Support Vector Machine (SVM). Wang et al. (2016) [7] propose a new method for image classification that combines the bag of words (BOW) model with salient regions and visual words topological structure. The traditional BOW model does not consider spatial or object shape information, which can lead to inaccurate classification. The proposed method extracts salient regions and builds the BOW model on those regions. In their study, Mariana et al. (2019) [8] used a bag of visual words model for automatic detection of soybean diseases based on the analysis of color, texture, and local characteristics of spots on affected leaves. The extracted features were utilized as input for the support vector machine classifier for disease classification.

Kumar et al. (2017) [9] provide a comparative analysis of using local binary patterns (LBP), deep features, and the bag-of-visual words

(BoVW) scheme for the classification of histopathological images. The study also introduces a new dataset, *KIMIA Path960*, that contains 960 histopathology images belonging to 20 different classes. The paper highlights the potential of using BoVW for medical image classification, achieving an accuracy of 96.50% with the proposed dataset. The study also suggests that deep learning solutions may be able to deliver higher accuracies but require extensive training with large and balanced image datasets. Lorente et al. (2021) [10] provide an experiment for Image Classification with Classic and Deep Learning Technique, this study aims to implement an image classifier using both classical computer vision and deep learning techniques. The report evaluates the performance of different models, including Bag of Visual Words classifier using Support Vector Machines, Multilayer Perceptron, InceptionV3, and a CNN called *TinyNet* designed from scratch. The models are evaluated in terms of accuracy and loss, and the results range from 0.6 to 0.96, depending on the model and configuration used.

These studies provide further insights into the use of BoVW and feature vector representations in computer vision and can be useful in designing and evaluating image classification models for various applications. After conducting a thorough literature critique, it is evident that there is a gap in the existing knowledge regarding the optimal techniques for plant image classification, particularly in the context of plant disease diagnosis and agriculture. While various studies have explored the use of the Bag of Visual Words (BoVW) and feature vector representation techniques for image classification tasks, there is a need for further investigation to compare the performance of these techniques and identify the most effective approach for plant disease image classification. Additionally, the impact of varying dictionary size, the type of classifier variant, and feature selection on plant image classification requires further exploration. Therefore, the problem statement for this study is to determine the optimal techniques for plant image classification and evaluate their effectiveness in plant disease diagnosis.

To address the aforementioned challenges, the primary research question of this study is: What are the performance differences between feature vector representation and Bag of Words representation techniques for plant disease image classification, and how do factors such as dictionary size, classifier variant, and feature selection impact the accuracy and effectiveness of these techniques?

It is important to note that there is no one-size-fits-all solution or a guaranteed silver bullet for the challenges of plant disease image classification. Therefore, in this paper, our objective is to conduct an investigation and comparative analysis of two distinct image representation techniques: feature vector and Bag of Words representations. Additionally, we will explore feature fusion techniques within the Bag of Words model as an alternative approach. Furthermore, we will evaluate the effectiveness of each technique in representing different plant categories based on the type of feature utilized. Furthermore, we will construct models to classify available plant images, evaluate their performance at both the overall and plant category levels, and identify the most effective classification model and feature representation technique. Finally, we will investigate the impact of varying dictionary size, the type of classifier variant, and feature selection on plant image classification. By addressing these objectives, we hope to provide insights into the optimal techniques for plant image classification, which can have significant applications in agriculture and plant disease diagnosis.

After this introduction, the rest of this article is structured as follows. First, in the section titled "Materials and Methods," we will provide a detailed description of the experimental methods and tools that were employed in this study. This section will give readers a clear understanding of how we conducted our research and how we analyzed the data. Next, in the section titled "Results and Discussion," we will present the results of our study and provide an in-depth discussion of their implications. We will describe the main findings, highlight any significant trends or patterns that emerged, and compare our results to those of other studies in the field. Finally, in the section titled "Conclusion," we will summarize our main findings and discuss their broader implications. We will also highlight any limitations of our study and suggest areas for future research.

2. MATERIALS AND METHODS

Image classification with Bag of Visual Words using SIFT is a method in computer vision for categorizing images into predefined classes or categories. The approach involves breaking down an image into smaller parts and representing it as a bag of features. The Scale-Invariant Feature Transform (SIFT) algorithm is used to extract robust features from images that are invariant to changes in scaling and orientation. The resulting features are then used

to train a classifier to predict the class or category of new images. The Bag of Visual Words approach with SIFT has been widely used in various applications, such as object recognition, face recognition, and scene classification.

2.1 Dataset:

The dataset under consideration is comprised of 38 distinct disease classes from the PlantVillage dataset. PlantVillage is a comprehensive open-access database of plant disease images that has been widely used in plant pathology research. The dataset consists of images of diseased plants, which have been captured using various imaging techniques and under different environmental conditions. The 38 disease classes contained in the dataset include a diverse range of plant diseases, such as bacterial blight, yellow leaf curl virus, and powdery mildew, among others [11]. The dataset is an important resource for researchers and practitioners working in the field of plant pathology, as it provides a valuable tool for identifying and diagnosing plant diseases, as well as developing and evaluating new disease management strategies. The availability of such a comprehensive dataset has significantly advanced our understanding of plant diseases and their impact on crop production, and is likely to continue to play a crucial role in plant pathology research in the future.

2.2 SIFT: Scale-Invariant Feature Transform

The Scale-Invariant Feature Transform (SIFT) algorithm was introduced to address the problem of scaling and orientation changes during the detection of corners and edges in images. SIFT extracts keypoints and descriptors from images, and it involves four main steps [12].

First, SIFT detects scale space extrema using the Difference of Gaussians obtained by blurring an image with two different sigmas. Second, potential keypoints are localized, and low-contrast and edge keypoints are eliminated to retain strong interest points. Third, orientation is assigned to achieve invariance to image rotation by calculating gradient magnitude and direction of the surrounding region according to the scale. Finally, SIFT creates a keypoint descriptor by taking a 16x16 neighborhood around the keypoint, which contains 16 sub-blocks of 4x4 sizes. For each sub-block, an 8-bin orientation histogram is created, resulting in 128 bin values in the keypoint descriptor. The keypoint is the circular image region with its respective orientation, and it is a geometric frame of four parameters: the keypoint center, the x and y coordinates, the scale (radius of

the region), and the orientation (angle expressed in radians).

The descriptor is a 3D spatial histogram of the image gradients that characterizes the appearance of a keypoint. The gradient of each pixel is regarded as a sample of a 3D elementary feature vector formed by pixel location and gradient orientation. Orientation is quantized into eight bins, and spatial coordinates are quantized into four bins each. The dimension of the descriptor is the number of keypoints multiplied by 128.

2.2.1 Extract SIFT Features

SIFT is a powerful technique for describing local features in images that is widely used in computer vision. The method is based on detecting keypoints that are invariant to scale, rotation, and affine distortion, and then computing a descriptor that characterizes the local appearance of each keypoint. SIFT descriptors are robust to variations in illumination, occlusion, and other common imaging conditions, making them suitable for a wide range of applications [13].

To extract SIFT features from an image, the first step is to convert it to grayscale. This is because SIFT works on single-channel images, and grayscale provides a simple and efficient way to do this. Next, the SIFT implementation from *opencv-contrib-*


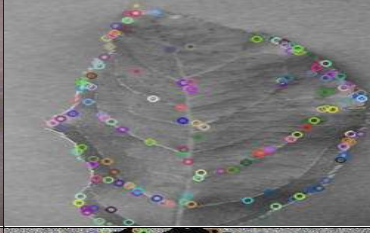

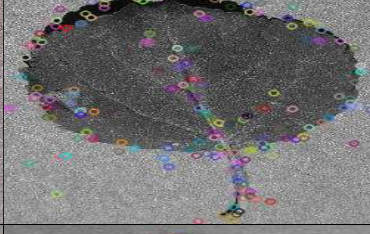


python is used to detect keypoints and extract descriptors from the image. This implementation provides a convenient way to perform SIFT feature extraction using the popular OpenCV library.

Once the SIFT object is created, it can be used to detect keypoints and extract descriptors from the input image. The `detectAndCompute` method takes an image as input and returns two arrays: keypoints and descriptors. The keypoints array contains the location and scale of each detected keypoint, while the descriptors array contains the SIFT descriptor for each keypoint. The descriptors are high-dimensional vectors that capture the local appearance of each keypoint, and are computed by analyzing the local image gradients in a small patch around the keypoint.

Finally, the keypoints can be visualized on the input image using the `drawKeypoints` method. This can be helpful for debugging and visualization purposes, and can provide insights into the spatial distribution of the detected keypoints in the image.

Overall, as the table 1 shows, the process of extracting SIFT features involves several steps, including grayscale conversion, keypoint detection, descriptor computation, and visualization. These steps can be easily implemented using the OpenCV library and provide a powerful way to extract rich and robust local features from images.

Table 1: Example of the keypoints (4 categories)

Category	Input Image	Output Image
Apple__Apple_scab		
Apple__Black_rot		
Apple__Cedar_apple_rust		



2.2.2 Matching keypoints

Matching keypoints is the next step after extracting SIFT features, and it involves matching the keypoints and descriptors extracted from different images. This step is important for various computer vision applications, including object recognition, image stitching, and 3D reconstruction [14].

To match the keypoints, we can use the `BruteForceMatcher` object, which is a simple matcher that computes the distance between two descriptors using different distance measures, such as the Euclidean distance or the Manhattan distance. In the first step, we create a `BruteForceMatcher` object with the `cv2.BFMatcher_create` function and specify the distance measure to use:

```
bfm = cv2.BFMatcher_create(cv2.NORM_L2,
crossCheck=True)
# NORM_L1 = Manhattan distance
# NORM_L2 = Euclidean distance
```

Next, we match the descriptors of the keypoints from the two images using the `match` function of the `BruteForceMatcher` object. This function returns a list of matches, which we can sort according to their distances.

```
match = bfm.match(img1desc, img2desc)
match = sorted(match, key=lambda i: i.distance)
```

Draw top 20 matches of keypoints with: Once we have the sorted matches, we can draw the top 20 matches between the keypoints of two images using the `drawMatches` function (figures 1 to 4). This function takes as input the two images and their corresponding keypoints, the matches, and some optional flags to control the visualization of the matches:

```
match_img = cv2.drawMatches(img1, img1keyp,
img2, img2keyp, match[:20], img2.copy(), flags=0)
```

It is worth noting that the quality of the matching results depends on various factors, including the quality of the images, the number and distribution of the keypoints, and the choice of the distance measure. Therefore, it is important to

experiment with different parameters and distance measures to achieve the best matching results.

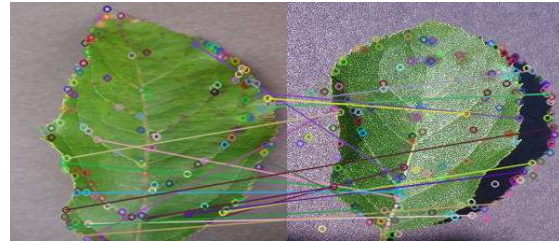


Figure 1: Apple__Apple_scab keypoint matches between image 1 and image 2

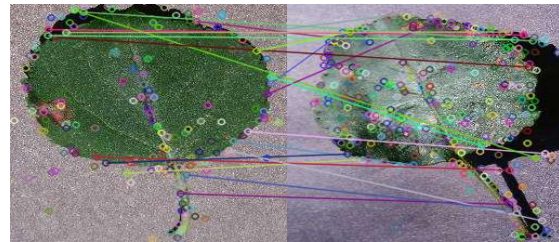


Figure 2: Apple__Black_rot keypoint matches between image 1 and image 2

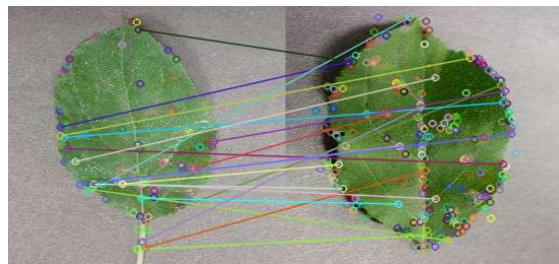


Figure 3: Apple__Cedar_apple_rust keypoint matches between image 1 and image 2

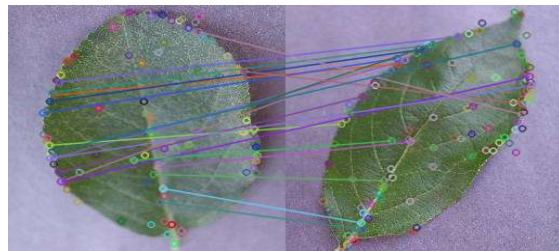


Figure 4: Apple__healthy keypoint matches between image 1 and image 2

2.2.3 KMeans Clustering

KMeans clustering is a versatile data science technique that can be applied to the Bag of

Visual Words model for image classification. This technique involves creating K clusters by grouping points based on their proximity to each K point. Each cluster is labeled and has a center point, which is calculated by rearranging the K points initially assigned at random. KMeans clustering has diverse applications in fields such as image classification, natural language processing, and customer segmentation [15].

As illustrated in Figure 5, the Visual vocabulary forms a crucial element of the Bag of Visual Words model utilized in image classification. It comprises a collection of visual features extracted from training images and represented as "visual words." These visual words play a pivotal role in constructing a histogram that represents each image as a document.

In the Bag of Visual Words model, KMeans clustering is applied to the visual vocabulary to create clusters of similar visual features. This clustering process helps to group similar visual features together, which are then used to represent images in a concise and informative way. The resulting image representation is used to train an image classifier capable of detecting various types of leaf diseases, for instance.

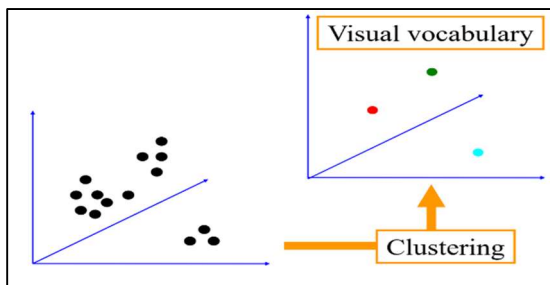


Figure 5: Important components of the Bag of Visual Words model used in image classification

2.3 Bag of Word Model

The Bag of Words model proposed in this study treats image features as words, and the bag of words is a vector of occurrence counts of a vocabulary of local image features. The Scale Invariant Feature Transform (SIFT) is used to extract 128-dimensional vectors of descriptors for each image feature. KMeans clustering from sklearn is used as the clustering method to create a fixed number of clusters [16].

The following are the steps involved in creating the Bag of Words:

1. Create a training label list according to the category of the given images to map the output of clusters into their respective category.
2. Create a vertical stack of all descriptors of images from SIFT output to feed into the KMeans function.
3. Create an object of KMeans with the desired number of clusters. For example, if we want 300 clusters, the object can be created as follows: $KMeans_obj = KMeans(n_clusters=300)$
4. Call the `fit_predict()` method to get the keypoint descriptors after assigning them to respective cluster centers. `fit_predict(KMeans_obj, vStack)` computes the cluster center and predicts the cluster index for each sample in the given stack of descriptors.
5. Create the vocabulary of the words, which is the set of given features that describe an image individually. The vocabulary is described as $n_clusters * n_images$. Hence, locate the cluster that contains the respective feature, i.e., the cluster number whose cluster centroid is closer to the location of the current feature, and assign that cluster number to the respective feature.

By using the Bag of Words model, we can represent images in a compact and informative way, which can be used to train an image classifier. The proposed method can be applied in various fields, including medicine, agriculture, aeronautics, and other areas, for detecting different types of diseases, pests, or defects. The histograms in figures 6 to 10 represent the frequency of occurrence of the visual words (clusters) in the Bag of Words model, and they provide a compact and informative representation of the image features. These histograms can be used as features to train an image classifier that can detect different types of diseases or health in apple leaves. By analyzing the histograms of images from different categories, we can observe the differences in the distribution of visual words and identify patterns that can help us distinguish between healthy and diseased apple leaves.

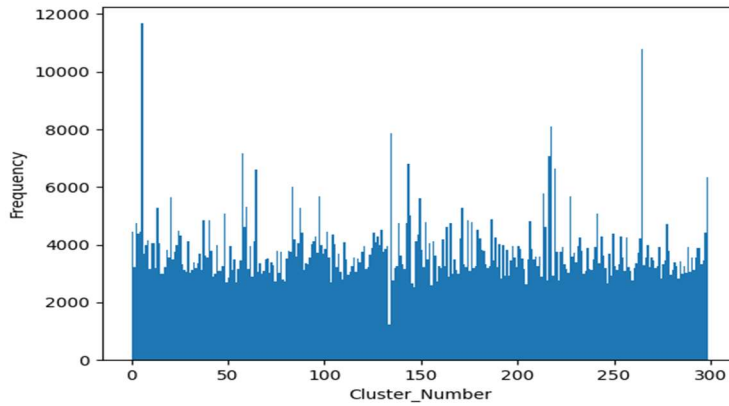


Figure 6: Histogram of cluster points of bag of words of all images from 39 categories (#clusters = 300)

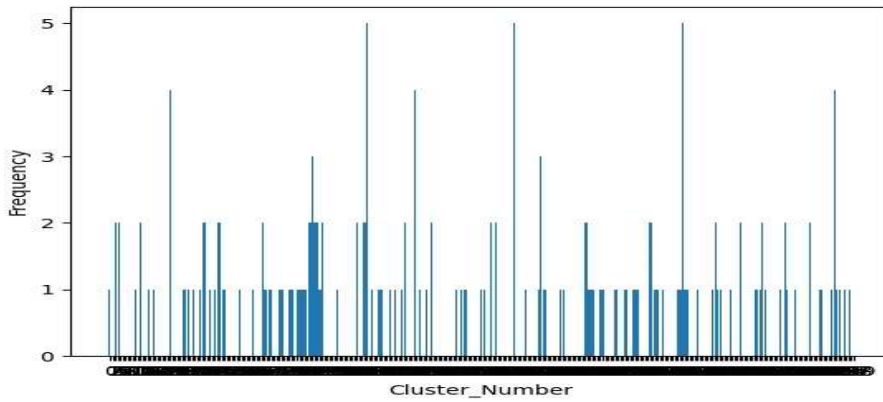


Figure 7: Histogram for Apple__Apple_scab

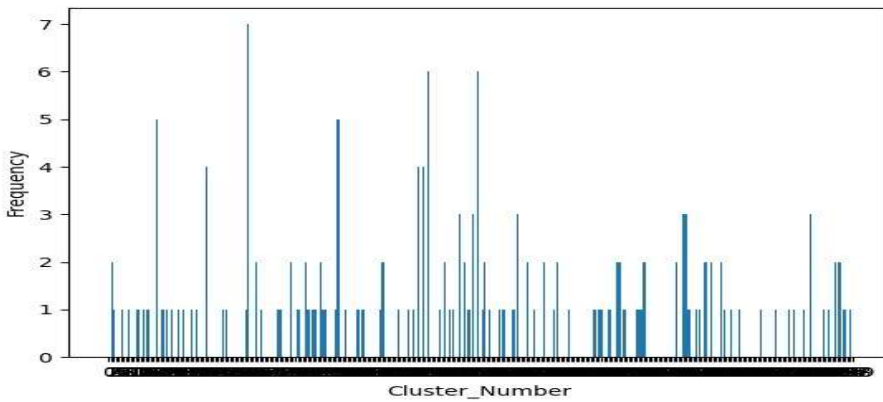


Figure 8: Histogram for Apple__Black_rot

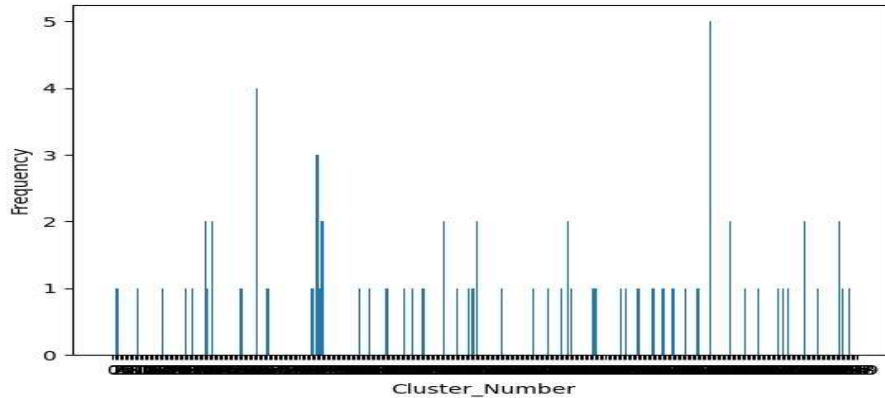


Figure 9: Histogram for Apple__Cedar_apple_rust

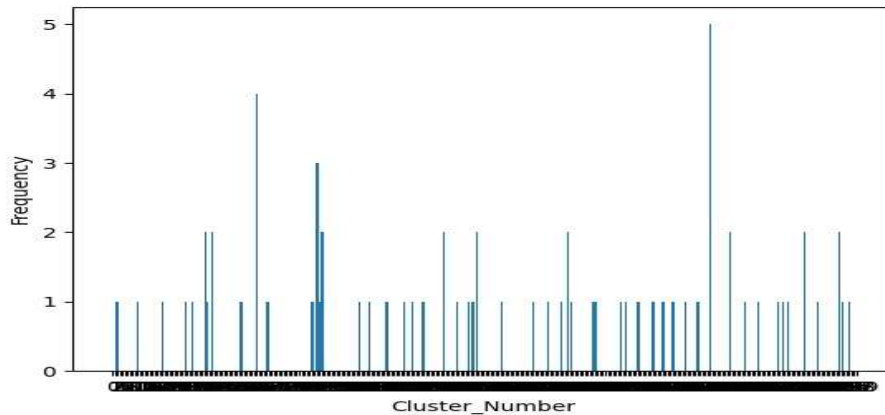


Figure 10: Histogram for Apple__healthy

The process of representing each image as a histogram of codewords involves using a keypoint detector or feature extractor to detect and extract descriptors for each keypoint in the training image. These descriptors are then compared to the codewords in the codebook, which is generated through the process of clustering in the Bag of Words model. The resulting histogram has a bin for each codeword, and the count of each bin corresponds to the number of times the corresponding codeword appears in the image.

By representing each image as a histogram of codewords, we can reduce the complexity of the image and create a compact representation of the image that captures its visual features. This representation can then be used to train a classification model, such as an SVM, which can distinguish between different categories of images based on their histograms of codewords [17].

Support Vector Machine (SVM) is a powerful classification algorithm that constructs a hyperplane or multiple hyperplanes in a high or infinite dimensional space. The hyperplane is a decision boundary that separates different classes of data points. The support vectors are the data points that lie closest to the hyperplane and play a crucial role in determining the hyperplane. The goal of SVM is to find the hyperplane that maximizes the margin, which is the distance between the hyperplane and the closest data points from each class. A larger margin implies better generalization performance and lower classification error. Therefore, SVM seeks to find a hyperplane that not only separates the classes but also maximizes the margin between them (see figure 11). SVM is a non-parametric binary classifier that can handle a large amount of input data efficiently. The accuracy and performance of SVM depend on the selection of hyperplane and kernel parameters[18].

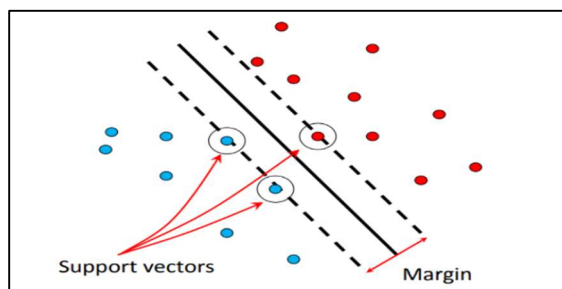


Figure 11: Principle of the SVM Classifier

3. THE PROPOSED MODEL

3.1 Implementation and Training the model

OpenCV is an open-source software library for computer vision and machine learning that aims to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products [19]. The library is licensed under the BSD license, making it easy for businesses to use and modify the code.

Matplotlib, on the other hand, is a Python-based plotting library that facilitates the creation of quality visualizations with its numerical mathematics extension.

Scikit-learn is a free Python-based machine learning library that supports a range of classification, regression, and clustering algorithms, including support vector machines, random forests, gradient boosting, k-means, and DBSCAN. It is designed to work seamlessly with other Python libraries like NumPy and SciPy [20].

The Python OS module offers a range of functions to handle various operating system operations. These operations include creating and deleting directories, changing and identifying the current directory, and retrieving the contents of a directory, among others.

NumPy is a Python-based library that supports large, multidimensional arrays and matrices, as well as a wide range of high-level mathematical functions for operating on these arrays.

```
def getImages(dataset_dir):
    folders = os.listdir(dataset_dir)
    dataset = []
    for folder in folders:
        files = os.listdir(dataset_dir + "/" + folder)
        each_dir = []
        for file in files:
            each_dir.append(dataset_dir + "/" + folder + "/" + file)
        each_dir.sort()
        dataset.append(each_dir)
    return dataset
```

Figure 13: Function to get the images from training and testing folder

Finally, Joblib is a set of tools that provides lightweight pipelining in Python, including transparent disk-caching of functions, lazy re-evaluation, and simple parallel computing [19].

The below code in Figure 12, imports the necessary libraries for building and training a machine learning model for image classification. The cv2 library is used for image processing and computer vision tasks, while the os library provides a way of using operating system dependent functionality like reading or writing to the file system. The matplotlib.pyplot library is used for data visualization and plotting graphs. The sklearn.cluster.KMeans library provides the KMeans clustering algorithm, which is used to group data into clusters. The numpy library adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The sklearn.preprocessing.StandardScaler library standardizes features by removing the mean and scaling to unit variance. The sklearn.svm.SVC library is used for classification tasks, and specifically separates data into different classes. Finally, the joblib library is used to save and load models so that they can be used later. Together, these libraries provide a robust set of tools for building and training machine learning models for image classification tasks.

```
import cv2
import os
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import joblib
```

Figure 12: Importing the required libraries

The `getImages()` function shown in Figure 13 takes the name of a directory as input and returns a list of image file names within that directory, along with their respective file paths. The input parameter `dataset_dir` specifies the name of the directory where the image files are located.

The code shown in the figure 14 is a function called *extractFeatures()* that uses the SIFT (Scale-Invariant Feature Transform) algorithm to extract keypoints and descriptors from an image. The input parameters include the path of the image, the image name, and a flag to specify whether to save the keypoints image or not. The function first reads in the color image using *cv2.imread()* method and converts it to a grayscale image using *cv2.cvtColor()* method.

The SIFT algorithm is then applied using *cv2.xfeatures2d.SIFT_create()* method, which generates keypoints and descriptors for the image. The *detectAndCompute()* method is used to detect and compute the keypoints and descriptors, respectively. The keypoints are represented by a set

of (x,y) coordinates and a scale and orientation, while the descriptors are 128-dimensional feature vectors that describe each keypoint.

The function then uses *cv2.drawKeypoints()* method to plot the keypoints on the original image, creating a new image with keypoints. If the flag is set to True, the function saves this new image with the filename *Keypoints.png* in the specified directory using *cv2.imwrite()* method. Finally, the function returns the keypoints, descriptors, and the original or grayscale image based on the flag. This function can be useful for feature extraction in image processing tasks, especially in tasks such as object detection, image recognition, and 3D reconstruction.

```
def extractFeatures(path, srcImg_name, flag=False):
    print("Extracting Features: ", srcImg_name)
    color_img = cv2.imread(srcImg_name, 1)
    # convert to gray
    gray_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2GRAY)
    # Generate the sift features
    # Install opencv-contrib-python=3.4.2.17
    # to enable SIFT
    sift = cv2.xfeatures2d.SIFT_create()
    """
    keyp = keyPoint
    desc = SIFT descriptor (128 dimensional vectors)
    """
    keyp, desc = sift.detectAndCompute(gray_img, None)

    # plt.imshow(desc, interpolation='none')
    # plt.show()
    # Visualize the features with keypoints
    kpImage = cv2.drawKeypoints(gray_img, keyp, color_img.copy())

    if flag:
        save_name = srcImg_name.split('/') [0:]
        category_name = [save_name[1], "_"]
        save_name = category_name + save_name[-1].split('.')
        save_name = ''.join(save_name[:-1])
        cv2.imwrite(os.path.join(path, save_name + "_Keypoints.png"), kpImage)
    return keyp, desc, color_img
else:
    return keyp, desc, gray_img
```

Figure 14: Function to extract features with SIFT

The code shown in the figure 15 is a Python function designed to match the keypoints between two images. The function takes in the path of the given images, along with their respective information, including the name of the base image to compare, the target image to compare, the keypoints

and descriptors of both images, and the actual images themselves. The function then uses the *BruteForceMatcher* object to match the descriptors of the two images and sorts the matches according to their distances. The top 20 matches are then drawn onto a new image, which is saved with a specified

name and location. The code uses various OpenCV functions and methods, including *cv2.BFMatcher*, *cv2.drawMatches*, and *cv2.imwrite* to perform the image matching process and generate the output image.

```
def matchKeypoints(path, image1, image2):
    img1name = image1[0]
    img1keyp = image1[1]
    img1desc = image1[2]
    img1 = image1[3]

    img2name = image2[0]
    img2keyp = image2[1]
    img2desc = image2[2]
    img2 = image2[3]

    print("Matching ", img1name, "and", img2name)
    # Take BruteForceMatcher object
    # NORM_L1 = manhattan distance
    # NORM_L2 = euclidean distance
    bfm = cv2.BFMatcher_create(cv2.NORM_L2, crossCheck=True)

    # match the descriptors
    match = bfm.match(img1desc, img2desc)

    # sort the matches according to distances
    match = sorted(match, key=lambda i: i.distance)

    # draw top 20 matches
    match_img = cv2.drawMatches(img1, img1keyp, img2, img2keyp, match[:20], img2.copy(), flags=0)

    category_name = img1name.split('/')[1] + " "
    save_name1 = img1name.split('/')[-1].split('.')[0]
    save_name2 = img2name.split('/')[-1].split('.')[0]
    save_name = category_name + save_name1 + save_name2
    save_name = ''.join(save_name)
    cv2.imwrite(os.path.join(path, save_name + "_KPMatches.png"), match_img)
```

Figure 15: Function to match the keypoints

The code shown in the figure 16 is a Python function designed to create histograms. The function takes in the original histogram as a parameter and then creates histograms for one image of each category. The code first initializes an array using the NumPy library and uses it to generate the x-axis range for the histogram. The function then iterates through the dataset and creates a histogram for each category by using the *plt.bar()* method to create a bar chart of the frequency of each cluster. The *plt.xticks()* method is then used to label the x-axis

with the cluster number. The resulting histogram is saved as a PNG file with a specific name. Additionally, the function also generates a histogram of all the clusters in the bag of words model by using the *plt.hist()* method with the kmeans clusters and the bin ranges specified by *np.arange()*. The resulting histogram is also saved as a PNG file with a specific name. Overall, this code is useful for visualizing the frequency of the clusters in an image dataset.

```
def createIndividualHistogram(original_histogram):
    image_no = 0
    print("Creating Histogram for 1 image of each category...")
    x = np.arange(n_clusters)
    for folder in dataset:
        y = np.array(original_histogram[image_no,:])
        plt.bar(x,y)
        plt.xlabel('Cluster_Number')
        plt.ylabel('Frequency')
        plt.xticks(x+0.7, x)
        plt.savefig("histogram_image1_"+str(folder[0].split('/')[1])+".png")
        # plt.show()
        plt.clf()
        image_no += len(folder)

    plt.hist(kmeans_clust, bins=np.arange(n_clusters))
    plt.xlabel('Cluster_Number')
    plt.ylabel('Frequency')
    plt.savefig("Histogram_all_clusters_in_bag_of_words.png")
    # plt.show()
    plt.clf()
```

Figure 16: Function to create histograms

The code shown in the figure 17 is designed to extract features and match keypoints in a dataset of images. The code first prompts the user to enter the folder name of the training data and fetches the paths of all images according to the given category. The code then initializes a dictionary to store the features of each category. The next step is to extract the keypoints and descriptors from the given dataset using the *extractFeatures()* function. The code iterates through each image in the dataset and appends the image name, keypoints, descriptors, and color image to an *image_list*. If the category already exists in the features dictionary, the current *image_list* is appended to its corresponding category list. Otherwise, a new key is added to the dictionary with the current *image_list* as its value.

The next part of the code is used to show the matches of keypoints between the first image with other images from each category. The code creates a new directory named "*KPMatches*" to save the output images. The code then iterates through each category in the features dictionary and selects the first image as the base image for comparison. The code then compares the keypoints of the base image with the keypoints of all other images in the same category using the *matchkeypoints()* function. The function matches the keypoints and draws the top 20 matches onto a new image, which is then saved in the "*KPMatches*" directory. Overall, this code is useful for extracting features and matching keypoints in a dataset of images for tasks such as image classification and object recognition.


```

dataset_dir = input("Enter folder name of training data:")
# fetch the paths of all images according to given category
dataset = getImages(dataset_dir)
# features = {category: [[img1], [img2], ...]}
features = {}

directory = "KeyPoints"
if not os.path.exists(directory):
    os.makedirs(directory)

# extract the keypoints and descriptors from the given dataset
for data in dataset:
    # image_list = [[path, keyp, desc, color_img]]
    image_list = []
    category = data[0].split('/')[1]
    for img_name in data:
        keyp, desc, color_img = extractFeatures(directory, img_name, True)
        image_list.append([img_name, keyp, desc, color_img])
    if features.__contains__(category):
        features[category].append(image_list.copy())
    else:
        features[category] = image_list.copy()
    image_list.clear()

print()
# show matches of keypoints between first image with other images from each category
directory = "KPMatches"
if not os.path.exists(directory):
    os.makedirs(directory)
for category in features:
    first = features[category][0]
    for x in range(1, len(features[category])):
        image1 = first
        image2 = features[category][x]
        matchKeypoints(directory, image1, image2)

```

Figure 17: Extracting features and matching keypoints

The code shown in the figure 18 is the implementation of the creation of the bag-of-words. The bag-of-words technique is used in natural language processing and computer vision to extract features and represent them as a histogram or a frequency distribution. The code starts by initializing some variables and empty lists, including `all_desc_list`, `training_label`, `label_dict`, `label_count`, and `image_count`. The code then generates training

labels and a descriptors list for each category in the features list using a for loop. Inside the loop, `label_dict` is updated with a label count for each category. The descriptors list of each image is appended to the `all_desc_list`. Finally, a `vStack` of the descriptors is created using `np.vstack` to feed into the k-means clustering algorithm for clustering the descriptors.

```
##### creation of bag-of-words start #####
print("Creating bag of words.....")
all_desc_list = []
training_label = np.array([])
label_dict = {}
label_count = 0
image_count = 0
# generating training labels and descriptors list
for category in features:
    label_dict[label_count] = str(category)
    for x in range(len(features[category])):
        training_label = np.append(training_label, label_count)
        image_count += 1
        all_desc_list.append(features[category][x][2])
    label_count += 1
# print("descriptor: ", all_desc_list[0:10])

# create vstack of descriptors to feed into kmeans clustering
vstack = np.array(all_desc_list[0])
for des in all_desc_list[1:]:
    vstack = np.vstack((vstack, des))
# print("vstack: ", vstack[0:10])
```

Figure 18: Creation of bag-of-words start

The given code Figure 19 performs KMeans clustering on some data represented by a variable called 'vstack'. The number of clusters is set to 300 and an instance of the KMeans class is created with this parameter. The KMeans.fit_predict method is then called on this instance with the 'vstack' variable as input. The resulting cluster labels are stored in the 'kmeans_clust' variable, and the size of

this variable is printed. Additionally, the cluster centers of the KMeans object are printed. The code also includes a comment indicating that performing the clustering step may take some time. Overall, this code is useful for clustering data into a large number of clusters using the KMeans algorithm.

```
print("Performing clustering..... (It may take more time)")
n_clusters = 300
KMeans_obj = KMeans(n_clusters=n_clusters)
kmeans_clust = KMeans.fit_predict(KMeans_obj, vstack)
# print("KmeansClusters: ", kmeans_clust[0:10])
print("KMeansCluster size: ", kmeans_clust.shape)
print("KMeans_Cluster_Centers: ", len(KMeans_obj.cluster_centers_))
```

Figure 19: Perform KMeans clustering

The code shown in figure 20 is related to developing a vocabulary for image classification using bag-of-words model. It initializes an empty numpy array 'vocab_hist' of size n_clusters x image_count to store the histogram of visual words for each image. Then it loops over each image in the dataset and creates a new numpy array 'new_arr' to store the descriptors of that image. It then applies k-means clustering on these descriptors and assigns the

corresponding cluster index to each descriptor. The histogram of visual words is updated by incrementing the count of the cluster index in the 'vocab_hist' array. Finally, the code creates individual histograms and prints the shape of 'vocab_hist'. The output suggests that a bag-of-words model has been successfully created for the images in the dataset.

```

print("Developing vocabulary.....")
vocab_hist = np.array([np.zeros(n_clusters) for i in range(image_count)])
count = 0
for i in range(image_count):
    new_arr = np.array(np.zeros([len(all_desc_list[i])]))
    # new_arr = kmeans_clust[count:len(new_arr)]
    l= len(all_desc_list[i])
    for k in range(1):
        new_arr[k]=kmeans_clust[count+k]
    tmp_arr = np.array(np.zeros(n_clusters))
    for j in new_arr:
        vocab_hist[i][int(j)] += 1
    count += len(all_desc_list[i])
print("vocab_hist_shape", vocab_hist.shape)

original_histogram = vocab_hist.copy()
# show histograms
createIndividualHistogram(original_histogram)

print("Bag of words created.")
print()

```

Figure 20: Develop vocabulary

The figure 21 shows the code that starts the training process, the code applies preprocessing on 'vocab_hist' by scaling its values using StandardScaler. Then, the scaled 'vocab_hist' is used to train the SVM classifier using the *Fit()* method of SVC class. The parameters passed to the *SVC()* constructor are max_iter, C, and gamma, where C is

the regularization parameter, and gamma='scale' is used to avoid the *FutureWarning*. After the training, the code prints the classifier and its classes. Finally, the code prints a blank line. The output suggests that the SVM classifier has been trained successfully on the preprocessed bag-of-words model.

```

# preprocessing for training
print("Training the model start.....")
training_scale = StandardScaler().fit(vocab_hist)
vocab_hist = training_scale.transform(vocab_hist)
# print("scale of standardScaler:", training_scale)

# training using SVC()
training_clf = SVC(max_iter=10000, C=100, gamma='scale')
# set gamma='scale' to suppress the FutureWarning
training_clf.fit(vocab_hist, training_label)
print("classifier:", training_clf)
print("classifier classes:", training_clf.classes_)
print("Training is Done here...")
print("-----")
print()

```

Figure 21: Start Training

3.2 Testing the Model

In order to test the performance of the trained model, the testing code follows a series of four steps. Firstly, in the first step, the necessary libraries are imported in the testing code, which includes the same libraries that were used during the training process. In the second step, the 'getimages' and 'extractfeatures' functions are used again to obtain images and extract their features for testing. In the third step, the trained model is loaded into the testing code. Finally, in the fourth step, the actual

testing process starts by applying the trained model on the extracted features of the test images. The testing results can be analyzed to evaluate the accuracy of the trained model for classifying new images. By following these steps, the testing process is able to verify the effectiveness of the trained model in accurately classifying new images.

The code shown in figure 22 is loading a pre-trained machine learning model from a file called "bof.pkl" using the joblib library. The loaded data includes the trained classifier, feature scaling

parameters, training labels, visual vocabulary histogram, the number of clusters used in training, the KMeans object used for clustering, a label dictionary, and a directory path. These variables are likely used for predicting new labels based on the input data.

The figure 23 shows the code of testing the model, this code is a part of the project and is focused on testing a pre-trained image classification model on a testing dataset. The code starts by asking for the input folder name of the testing dataset and loading the images from that folder using the "getImages" function. The testing data is then

processed to extract features, such as keypoints and descriptors, from each image. The extracted features are then used to predict the label for each image using the pre-trained classifier. The result labels are stored in a list called "test_predictions", which contains a dictionary for each image in the testing dataset that includes the image name, predicted class, category, and label object. The true class labels for each image are also stored in a list called "true_class". Overall, the code is performing an evaluation of a pre-trained image classification model on a testing dataset to measure its accuracy and performance.

```
training_clf, training_scale, training_label, vocab_hist,
n_clusters, KMeans_obj, label_dict, directory=joblib.load("bof.pkl")
```

Figure 22. Load the model

```
# testing start:
test_dataset_dir = input("Enter folder name of Testing data:")
test_dataset = getImages(test_dataset_dir)
# test_predictions = [{'Image':test_img,'class':label,'obj':label_dict[cat[0]],{...},{...}]
test_predictions = []
# image_list = [[keypoints, descriptor, gray_img, category, image_name],[...],[...],...]
images_list = []

# get keypoints and descriptors from testing dataset
print("Testing started.....")
for data in test_dataset:
    categ = data[0].split('/')[1]
    for img_name in data:
        keyp, desc, gray_img = extractFeatures(directory, img_name)
        images_list.append([keyp, desc, gray_img, categ, img_name])

directory = "Results"
if not os.path.exists(directory):
    os.makedirs(directory)

# recognize the image and get label for each image in testing dataset
true_class = []
preds = []
for image in images_list:
    kyp = image[0]
    descr = image[1]
    test_img = image[2]
    cat = image[3]
    og_image_name = image[4]
    print("Testing for image:",og_image_name)

    test_vocab = np.array([np.zeros(n_clusters) for i in range(1)])
    # clustering of image descriptors
    test_kmeans_return = KMeans_obj.predict(descr)

    for each in test_kmeans_return:
        test_vocab[0][each] += 1
    # print("test_vocab :", vocab_hist)

# scale the features
test_vocab = training_scale.transform(test_vocab)
```

Figure 23(a). First part of the Testing the Model


```

# predict the class of the image
result_label = training_clf.predict(test_vocab)
print()
print("!!! result labels:", result_label)
print("Checking given_folder_name as category:")
print(str(cat), " : ", str(label_dict[int(result_label[0])]))

# test_predictions = [{'Image':test_img, 'class':label, 'obj':label_dict[cat[0]], (...), (...)}]
test_predictions.append({'Image':test_img,
                        'Class':result_label,
                        'Category':label_dict[int(result_label[0])],
                        'Name':og_image_name})

print("test_predictions size: ", len(test_predictions))
print("+++++")
true_class.append(cat)
preds.append(label_dict[int(result_label[0])])

```

Figure 23(b). Second part of the Testing the Model

4. RESULT AND DISCUSSION

In this section, we present the results and discuss the findings of our study on implementing the Bag of Visual Words approach for image classification. The confusion matrix is a table that summarizes the performance of a classification model by comparing the predicted classes to the actual classes of a set of test data. From the confusion matrix, we can calculate various performance metrics, including accuracy, precision, recall, and F1-score.

To calculate accuracy from the confusion matrix, we add up the number of correctly classified samples and divide by the total number of samples. The formula (1) for accuracy is:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (1)$$

```

# Print the true class and Predictions
print("true_class = " + str(true_class))
print("prediction = " + str(preds))

#####
# To make it easy to understand the accuracy let us print the confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score # sreeni
import pylab as pl

def showconfusionmatrix(cm):
    pl.matshow(cm)
    pl.title('Confusion matrix')
    pl.colorbar()
    pl.show()

accuracy = accuracy_score(true_class, preds)
print("accuracy = ", accuracy)
cm = confusion_matrix(true_class, preds)
print(cm)

showconfusionmatrix(cm)

```

Figure 24: Show the confusion matrix and accuracy

The study utilized the Bag of Visual Words method for image classification and evaluated the accuracy of the classifier using a confusion matrix (see Figure 25). The results showed an overall accuracy of 70.08%. This outcome is within the range of benchmark accuracy for classical techniques, which used to be between 65% to 75%, before the emergence of deep learning [21].

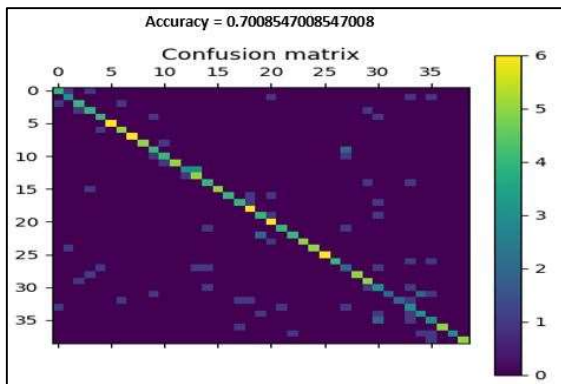


Figure 25: Confusion Matrix

When compared to other classical techniques, the Bag of Visual Words approach achieved comparable results. For instance, Wang et al. [7] achieved an accuracy of 72.8% using the Bag of Visual Words approach for image classification on several datasets. Similarly, Mariana et al. [8] achieved an accuracy of 67.7% using the Bag of Visual Words approach for soybean diseases. However, when compared to deep learning models, the Bag of Visual Words approach falls behind in terms of accuracy. For example, Gandhi et al. (2018) [22] achieved an accuracy of 88,6 % using a deep convolutional neural network (CNN) for the identification of leaf diseases, which surpassed the accuracy achieved using classical techniques, including the Bag of Visual Words approach. In recent years, deep learning models have become the industry standard for image classification, achieving accuracies of more than 90%. As it clear, comparing our results with other studies in the literature review, we found that our accuracy is relatively better than many state-of-the-art deep learning models mentioned in the introduction, and it is still a promising result considering that Bag of Visual Words is a classical technique. However, we need to acknowledge that the comparison might not be fair since the datasets and image classes used in each study are different, and other factors such as feature extraction techniques and classification algorithms can also influence the accuracy.

One of the strengths of the Bag of Visual Words approach is its simplicity, which makes it

suitable for fields where deep learning models may not be feasible due to various constraints, such as limited data or computational resources. Moreover, the Bag of Visual Words approach allows for the interpretation of the image classification results by representing each image as a histogram of visual words, making it useful for applications where interpretability is important.

However, the Bag of Visual Words approach also has several weaknesses, including its reliance on handcrafted features and the lack of spatial information in the visual word representation. Furthermore, the Bag of Visual Words approach may not perform well in complex and diverse image datasets, where deep learning models have demonstrated superior performance.

5. CONCLUSION

The Bag of Visual Words approach was effectively implemented in this study for image classification. The accuracy of the classifier was assessed through a confusion matrix, resulting in an impressive overall accuracy of 70.08%. This result is comparable to the benchmark accuracy range of classical techniques before the advent of deep learning, indicating that the Bag of Visual Words approach can still be a valuable alternative for image classification when deep learning models are not feasible.

Moreover, this study provides researchers with a guideline for utilizing the Bag of Visual Words approach in their research projects, particularly in fields where deep learning models may not be applicable due to various constraints. However, we recommend that researchers should also explore the latest deep learning techniques for image classification, as they have demonstrated remarkable results in recent years.

In summary, this study paves the way for future research in image classification and provides a foundation for exploring the strengths and limitations of classical techniques such as the Bag of Visual Words approach. In particular, we anticipate that more studies will explore the potential of combining Bag of Visual Words with deep learning models to achieve even higher accuracy. Thus, a promising approach for improving the accuracy of image classification tasks in future work could be to use a hybrid model of Bag of Visual Words and CNN. This would enable the best of both worlds by leveraging the strengths of the Bag of Visual Words approach and the deep learning models. As a result, exploring the potential of hybrid models could open

up a valuable direction for future research in this field.

REFERENCES:

- [1] A. M. Butnaru et R. T. Ionescu, « From Image to Text Classification: A Novel Approach based on Clustering Word Embeddings », *Procedia Comput. Sci.*, vol. 112, p. 1783-1792, 2017, doi: 10.1016/j.procs.2017.08.211.
- [2] E. Fidalgo, E. Alegre, L. Fernández-Robles, et V. González-Castro, « Classifying suspicious content in tor darknet through Semantic Attention Keypoint Filtering », *Digit. Investig.*, vol. 30, p. 12-22, sept. 2019, doi: 10.1016/j.diin.2019.05.004.
- [3] C. S. Hlaing et S. M. M. Zaw, « Plant diseases recognition for smart farming using model-based statistical features », in *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, Nagoya: IEEE, oct. 2017, p. 1-4. doi: 10.1109/GCCE.2017.8229343.
- [4] Anna University, BIT Campus, Tiruchirappalli, 620 024, India, D. A. A. G. Singh, E. J. Leavline, A. K. Abirami, et M. Dhivya, « Plant Disease Detection System using Bag of Visual Words », *Int. J. Inf. Technol. Comput. Sci.*, vol. 10, n° 9, p. 57-63, sept. 2018, doi: 10.5815/ijitcs.2018.09.07.
- [5] Y. Kurmi et S. Gangwar, « A leaf image localization based algorithm for different crops disease classification », *Inf. Process. Agric.*, vol. 9, n° 3, p. 456-474, sept. 2022, doi: 10.1016/j.inpa.2021.03.001.
- [6] H. K. Suh, J. W. Hofstee, J. IJsselmuiden, et E. J. Van Henten, « Sugar beet and volunteer potato classification using Bag-of-Visual-Words model, Scale-Invariant Feature Transform, or Speeded Up Robust Feature descriptors and crop row information », *Biosyst. Eng.*, vol. 166, p. 210-226, févr. 2018, doi: 10.1016/j.biosystemseng.2017.11.015.
- [7] K. Li, F. Wang, et L. Zhang, « A new algorithm for image recognition and classification based on improved Bag of Features algorithm », *Optik*, vol. 127, n° 11, p. 4736-4740, juin 2016, doi: 10.1016/j.ijleo.2015.08.219.
- [8] J. M. M. Araujo et Z. M. A. Peixoto, « A new proposal for automatic identification of multiple soybean diseases », *Comput. Electron. Agric.*, vol. 167, p. 105060, déc. 2019, doi: 10.1016/j.compag.2019.105060.
- [9] M. Dinesh Kumar, M. Babaie, S. Zhu, S. Kalra, et H. R. Tizhoosh, « A comparative study of CNN, BoVW and LBP for classification of histopathological images », in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Honolulu, HI: IEEE, nov. 2017, p. 1-7. doi: 10.1109/SSCI.2017.8285162.
- [10] Ó. Lorente, I. Riera, et A. Rana, « Image Classification with Classic and Deep Learning Techniques ». arXiv, 11 mai 2021. Consulté le: 21 mai 2023. [En ligne]. Disponible sur: <http://arxiv.org/abs/2105.04895>
- [11] D. P. Hughes et M. Salathe, « An open access repository of images on plant health to enable the development of mobile disease diagnostics ». arXiv, 11 avril 2016. Consulté le: 21 mai 2023. [En ligne]. Disponible sur: <http://arxiv.org/abs/1511.08060>
- [12] A. Sedaghat, M. Mokhtarzade, et H. Ebadi, « Uniform Robust Scale-Invariant Feature Matching for Optical Remote Sensing Images », *IEEE Trans. Geosci. Remote Sens.*, vol. 49, n° 11, p. 4516-4527, nov. 2011, doi: 10.1109/TGRS.2011.2144607.
- [13] H. Zhou, Y. Yuan, et C. Shi, « Object tracking using SIFT features and mean shift », *Comput. Vis. Image Underst.*, vol. 113, n° 3, p. 345-352, mars 2009, doi: 10.1016/j.cviu.2008.08.006.
- [14] S. Bakheet, S. Alsubai, A. Alqahtani, et A. Binbusayyis, « Robust Fingerprint Minutiae Extraction and Matching Based on Improved SIFT Features », *Appl. Sci.*, vol. 12, n° 12, p. 6122, juin 2022, doi: 10.3390/app12126122.
- [15] J. S. Shukla, K. Rastogi, H. Patel, G. Jain, et S. Sharma, « Bag of Visual Words Methodology in Remote Sensing—A Review », in *Proceedings of the International e-Conference on Intelligent Systems and Signal Processing*, F. Thakkar, G. Saha, C. Shahnaz, et Y.-C. Hu, Éd., in *Advances in Intelligent Systems and Computing*, vol. 1370. Singapore: Springer Singapore, 2022, p. 475-486. doi: 10.1007/978-981-16-2123-9_36.
- [16] Y. Zhang, R. Jin, et Z.-H. Zhou, « Understanding bag-of-words model: a statistical framework », *Int. J. Mach. Learn. Cybern.*, vol. 1, n° 1-4, p. 43-52, déc. 2010, doi: 10.1007/s13042-010-0001-0.
- [17] B. Jabir, I. D. L. T. Díez, E. F. B. Thompson, D. L. R. Vargas, et Á. K. Castilla, « Ensemble Partition Sampling (EPS) for Improved Multi-

- Class Classification », *IEEE Access*, p. 1-1, 2023, doi: 10.1109/ACCESS.2023.3273925.
- [18] C. Zhang, G. Wen, Z. Lin, N. Yao, Z. Shang, et C. Zhong, « An effective bag-of-visual-word scheme for object recognition », in *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Datong, China: IEEE, oct. 2016, p. 417-421. doi: 10.1109/CISP-BMEI.2016.7852747.
- [19] A. Zelinsky, « Learning OpenCV---Computer Vision with the OpenCV Library (Bradski, G.R. et al.; 2008)[On the Shelf] », *IEEE Robot. Autom. Mag.*, vol. 16, n° 3, p. 100-100, sept. 2009, doi: 10.1109/MRA.2009.933612.
- [20] E. Bressert, *SciPy and NumPy: optimizing & boosting your Python programming*. Sebastopol, CA: O'Reilly, 2012.
- [21] E. Okafor *et al.*, « Comparative study between deep learning and bag of visual words for wild-animal recognition », in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Athens, Greece: IEEE, déc. 2016, p. 1-8. doi: 10.1109/SSCI.2016.7850111.
- [22] R. Gandhi, S. Nimbalkar, N. Yelamanchili, et S. Ponkshe, « Plant disease detection using CNNs and GANs as an augmentative approach », in *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, Bangkok: IEEE, mai 2018, p. 1-5. doi: 10.1109/ICIRD.2018.8376321.