Leaf Segmentation

Introduction to Image Processing and Computer Vision - Project 2
Report

Noman Noor

January 2023

Contents

1 Introduction

Techniques of Image Processing and Computer Vision when coupled with Artificial Intelligence can prove to be useful for a variety of problems such as recognition, classification, etc. The goal of this project is to solve one such problem, namely Classification, using various *features* of images.

2 Problem Description

2.1 Dataset Description

The dataset used for the project is the KTH-texture-data. The dataset contains images of 11 different types/classes of textures such as Aluminium foil, Cotton, etc., and is split into "Train" and "Valid" parts. Train is to be used for training the model, and Valid is used to test and validate the accuracy of the trained model.

The dataset has total of 3195 images of textures, and 1595 of those are from the "Train" part and 1600 are from the "Valid" part. Each of these parts contain images of various textures. The following is the list of all the types of textures contained in each of the parts:

(a) Aluminium Foil

(b) Brown Bread

(c) Corduroy

(d) Cork

(e) Cotton

(f) Cracker

(g) Linen

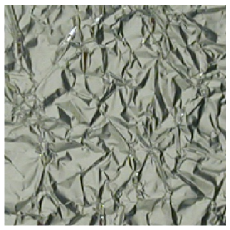
(h) Orange Peel

(i) Sponge

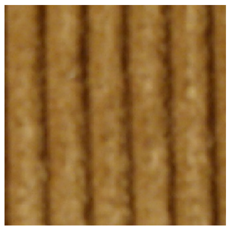
(j) Styrofoam

(k) Wool

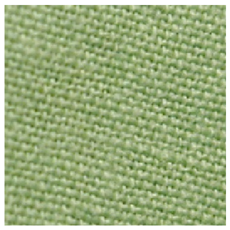
It is to be noted though, that number of images of each type of texture are not equal in any of the parts of the dataset. As such, training might be skewed to some extent.


(a) Aluminium Foil

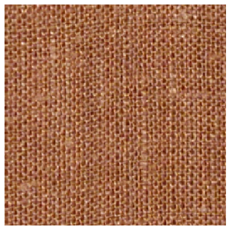
(b) Brown Bread


(c) Corduroy

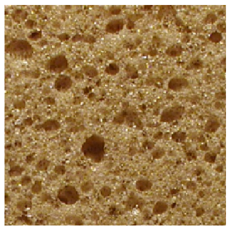
(d) Cork

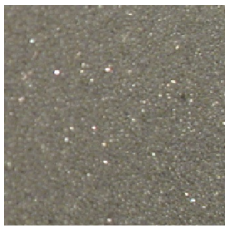

(e) Cotton

(f) Cracker


(g) Linen

(h) Orange Peel


(i) Sponge


(j) Styrofoam

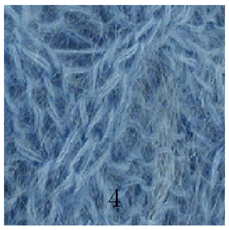
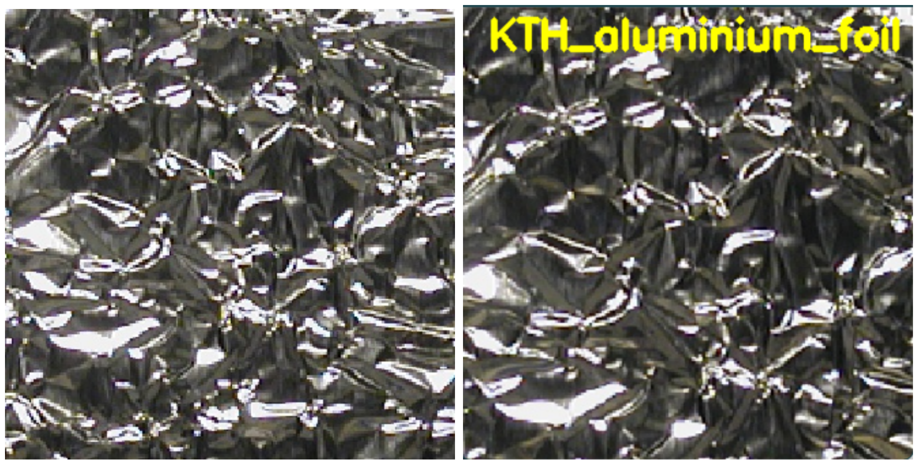

(k) Wool

Figure 1: Examples of each type of texture from the dataset.

2.2 Task Description

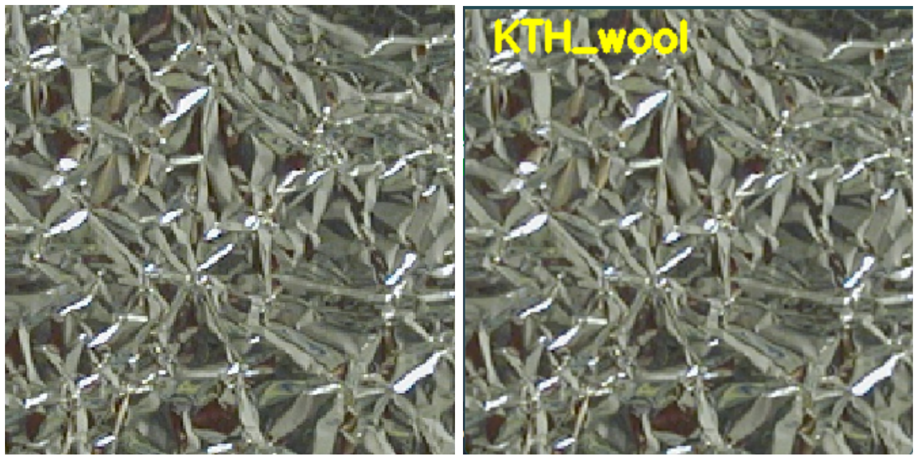
The goal of the project is to train a model on various different local or global features and then using it on the validation set to compare the accuracy of those features in classification of the textures.
For example, if one were to make a prediction on a picture of aluminum foil, and the model classifies it accurately, one should see the following results:



(a) Input Aluminium Foil (b) Prediction

Figure 2: An example of correct/accurate classification.

As for inaccurate classifications, the following visualisation serves as an example:



(a) Input Aluminium Foil (b) Prediction

Figure 3: An example of incorrect/inaccurate classification.

3 Description of Solution

3.1 Loading the Dataset and Preprocessing

The first step is loading the dataset. Due to the dataset directory structure, the labels for training are the names of the folders containing each type of image (e.g. KTH_wool) as it is more convenient but also self-explanatory. First, the images are read into 2D arrays using OpenCV, because that's the format the images need to be in for now.
Next, as soon as each image is loaded, the images need to go through some pre-processing. Due to the quality of the dataset being decent, the only type of pre-processing that was deemed to be needed was converting the images into grayscale as that is the format needed for some of the next steps.

3.2 Feature Extraction

Having read the images, in order to proceed further to the training stage, a feature vector for is needed for each image. As such, the step after preprocessing is extraction of features.
There are various types of features that can be extracted for the project as the goal of this project is the comparison of these features' efficacy in classification. The application provides methods for extraction of any of these features in the `ExtractionHelper` class in the `featureextractionhelper.py` file:

Global Features

1. **Haralick** - These are texture features, based on the adjacency matrix (the adjacency matrix stores in position (i,j) the number of times that a pixel

takes the value i next to a pixel with the value j. Given different ways to define next to, you obtain slightly different variations of the features. Standard practice is to average them out across the directions to get some rotational invariance.

2. **Local binary patterns (LBP)** - are a more recent set of features, in which each pixel is looked at individually. Its neighbourhood is analysed and summarised by a single numeric code. The normalised histogram across all the pixels in the image is the final set of features. These specifically attempt at capturing texture. LBPs are insensitive to orientation and to illumination (scaling).

3. **Threshold Adjacency Statistics (TAS)** - These are a recent innovation too, however, not much could be found in the way of documentation on these.

4. **Zernike Moments** - Zernike moments are not a texture feature, but rather a global measure of how the mass is distributed. These were included regardless of that in order to show how much worse these are in the given scenario.

5. **Gray Level Coocurrence Matrix (GLCM)** - The GLCM functions characterize the texture of an image by calculating how often pairs of pixel with specific values and in a specified spatial relationship occur in an image, creating a GLCM, and then extracting statistical measures from this matrix. These statistics provide information about the texture of an image.

Local Features

1. **Speeded-Up Robust Features (SURF)** - it is a patented local feature detector and descriptor, and can be used for tasks such as object recognition, image registration, classification, or 3D reconstruction. It is partly inspired by the scale-invariant feature transform (SIFT) descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT.

Depending on the starting configuration of the application (value of the `extract_features` variable), any one of the above feature vectors can be used, but only one type may be used during a single run of the program.

3.3 Classifier and training

Since it was made clear in the task that the classifier chosen was irrelevant to the current purpose of the task, the relatively simple classifier, **Linear Support Vector Classifier (Linear SVC)** was chosen. The objective of a Linear SVC (Support Vector Classifier) is to fit to the data provided to it, returning a "best fit" hyperplane that divides, or categorizes, the said data. From there, after getting the hyperplane. Following that, some features can be inputted to the classifier to see what the "predicted" class is.

After loading and initializing Linear SVC, it is provided with the training data and their labels. Then, training of the model begins by calling the *fit* method/function.

3.4 Validation and Accuracy

After training, the model is used to predict the labels of the test/validation set ("Valid"). Then, the `‘‘accuracy_score"` function from the `scikit-learn` library is used to check the accuracy of the predictions. The function measures the accuracy based on the number of inputs where the predicted label (texture class in this case) is the same as the actual label.

4 Assessment of the Results

While there is an option for users to show the predictions for each each image (by setting the "isShowPredictionsEnabled" variable to "True"), and as such humans could judge these images and check which traits that are judged correctly and which are judged incorrectly, in this case it makes more sense to check the accuracy using known correct labels against the predicted labels as described in section 3.4.

The following are the accuracy figures for each of the different feature vectors described in section 3.2:

	LBP	**Haralick**	**SURF**	**TAS**	**Zernike**	**GLCM**
Accuracy	0.710625	0.485625	0.64125	0.693125	0.295625	0.415625

It might be of note that the classifier used was Local Support Vector Classifier as described in section 3.3

It can be observed that LBP is the most accurate for the given dataset, and it makes sense since its main purpose is for textures. On the contrary, GLCM is also supposed to be good for textures, but doesn't perform well for the given dataset. Meanwhile, Zernike Moments is supposed to be bad at textures, and it can be seen that it is true from its result as the worst performing feature vector.

5 Potential Improvements to the Program

One could envision various improvements to the current solution, including the following.

- Adding more feature vector choices.
- Choosing a more well-suited and complex classifier.

- Combining some of the feature vectors and showing the results of them as well.
- Potentially including some other forms of preprocessing that might help with the results.

6 Conclusion

While none of the feature vectors combined with the chosen classifier yield results that are accurate enough, LBP does certainly come very close. In fact, even some potential improvements have been identified. However, the purpose of the project was comparing different feature vectors, and that has been achieved. The assessments of the results enabled comparison between various feature vectors. For example, for the given dataset and the chosen classifier combination, it was determined that LBP performed the best. Meanwhile, Zernike Moments performed worse than expected. The assessment served as a good comparison point, which was the purpose of the project. As such, it can be said this solution to the project is successful. However, it bears repeating that none of the solutions tested so far have been as accurate as one might hope for.

Regardless of the issues and problems, or even the accuracy of results, the program achieves its main goal of comparing the efficacy of various feature vectors (for the given dataset and the chosen classifier) fairly well.