# Implementing Fast GLCM for Classification of Remote Sensing Images

Deepankar Sehra
*Dept. of Electrical Engineering*
IIT Bombay
20d070024@iitb.ac.in

Abheet Chaudhary
*Dept. of Electrical Engineering*
IIT Bombay
20d070002@iitb.ac.in

Samriddhi Mishra
*Dept. of Electrical Engineering*
IIT Bombay
20d070068@iitb.ac.in

***Abstract*—This paper presents a method for implementing Fast Gray-Level Co-occurrence Matrix (GLCM) on remote sensing images. The GLCM technique is utilized to extract texture features from an image by capturing the spatial relationship between pairs of pixels. In this work, we implemented a fast GLCM algorithm that utilizes a sliding window approach to compute the co-occurrence matrix for each pixel in the image, allowing for more efficient handling of large images.**

***Index Terms*—Classification; fast GLCM; remote sensing; texture feature.**

## I. Introduction

Remote sensing images play an important role in various fields, such as agriculture, forestry, and environmental monitoring. Extracting features from these images is an important task that can be used to classify and analyze these images. Out of many available techniques one of the most effective for feature extraction is the Gray-Level Co-occurrence Matrix (GLCM), which captures the spatial relationship between pixel intensities. Today GLCM is widely used in various fields including medical imaging, remote sensing, and materials science, for image classification, segmentation, and texture analysis. However, the computation time required for traditional GLCM implementations can be a challenge for large remote sensing images.

To overcome this, we implemented a fast GLCM algorithm that reduces the computation time while maintaining the accuracy. In this paper, we present the details of our approach, including the GUI we developed for users to interact with the algorithm.

First, we provide an overview of GLCM and fast GLCM, explaining how they work and highlighting the differences between the two techniques. Next, we describe our methodology, including the algorithm we developed, the implementation details, and the data used for testing and evaluation. We then present the GUI, which allows users to customize the parameters for GLCM feature extraction and classification.

Our experiments demonstrate that our fast GLCM algorithm provides significant improvements in computation time while maintaining comparable accuracy to traditional GLCM. Finally, we discuss the limitations of current implementations of fast GLCM. Overall, this paper provides a comprehensive overview of our approach.

## II. What is GLCM and Fast-GLCM

Gray-Level Co-occurrence Matrix (GLCM) is a technique used in image processing and computer vision to extract texture features from an image. It captures the spatial relationship between pairs of pixels in an image by counting the number of times a pair of pixels with specific intensity values and at a particular spatial offset occurs in an image. This produces a matrix that contains information about the frequency and distribution of pixel pairs in the image. In GLCM, numerical features were extracted from the image in terms of the second-order statistics and they provide a measure of the underlying statistics.

Gray level Co-occurrence Matrices capture properties of a texture but they are not directly useful for further analysis, such as the comparison of two textures. Numeric features are computed from the co- occurrence matrix that can be used to represent the texture more compactly.

The conventional GLCM algorithm generates a co-occurrence matrix by comparing each pixel of an image with its neighboring pixel in a designated direction and distance. This process is repeated for all directions and distances, leading to a substantial co-occurrence matrix that characterizes the spatial correlation between all pixel pairs in the image. In comparison, the Fast GLCM employs a sliding window technique to compute the co-occurrence matrix for every pixel in the image, which enables it to handle extensive images more effectively. Significant savings possible when window size is large, and many features are computed.

## III. Methodology and Approach

Our target is to write a Fast GLCM code so that we can implemet it on remote sensing images and extract the features from the images and classifying accordingly. We have also created a GUI which allows the user to add options for selection in order to make things more user friendly.

Here is our approach in its simplest form:

- Take a window of size **w**
- For all pixels in the window compute the GLCM matrix
- Derive the features from the matrix
- Slide the window by 1 pixel horizontally or vertically
- Repeat the process until all pixels are covered
- Give the extracted features to the k-means clustering algorithm
- Classify the pixels of an image

There are already library functions available to do these things, but to get a better understanding of what we learned, we tried to implement most of these steps ourselves.

### *A. Definition of GLCM*

The Gray-Level Co-occurrence Matrix (GLCM) is a two-dimensional array, where each row and column represents a possible image value. GLCM is constructed by counting all pairs of pixels with gray levels **i** and **j**, separated by a displacement vector **d**=(dx, dy), which determines the directionality of texture. For instance, a horizontal direction is represented by dx=1, dy=0, and a diagonal direction by dx=1, dy=1.

$$P_d(i,j) = n_{i,j} = \#\{f(m,n) = i, f(m+dx, n+dy) = j; \\ 1 \leq m \leq M; 1 \leq n \leq N\}$$

where $n_{ij}$ is the number of occurrences of the pixel values **(i, j)** lying at a distance **d** in the image. The co-occurrence matrix $P_d$ has dimension **n x n**, where **n** is the number of gray levels in the image.

### *B. Algorithm to construct GLCM*

Count all pairs of pixels in which the first pixel has a value **i**, and its matching pair displaced from the first pixel by **d** has a value of **j**

This count is entered in the $i^{th}$ row and $j^{th}$ column of the matrix $P_d[i,j]$.

Note that $P_d[i,j]$ is not symmetric in this form of counting, since the number of pairs of pixels having gray levels [i,j] does not necessarily equal the number of pixel pairs having gray levels [j,i].

The elements of $P_d[i,j]$ can be normalized by dividing each entry by the total number of pixel pairs. Normalized GLCM $N[i,j]$, defined by:

$$N[i,j] = \frac{P[i,j]}{\sum_i \sum_j P[i,j]}$$

it normalizes the co-occurrence values to lie between 0 and 1, and allows them to be thought of as probabilities.

### *C. Fast Computation of GLCM*

To increase the computation speed we can use the fact that the number of pixel pairs that are common to computation of GLCM/features at successive position of the window remain similar. When texture window moves by 1 pixel right, the first column moves out of the computation while the last column enters the computation thus many pixels remain unchanged.

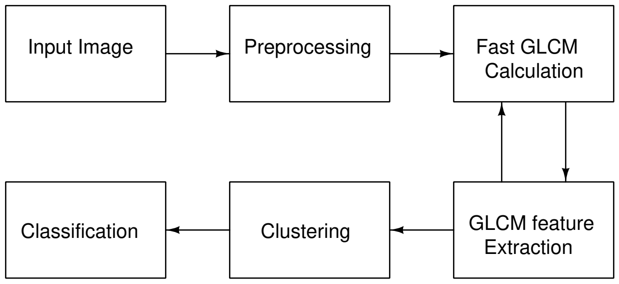

Fig. 1. Workflow of the algorithm

### *D. Numeric Features from GLCM*

Gray level co-occurrence matrices capture properties of a texture but they are not directly useful for further analysis, such as the comparison of two textures. Numeric features are computed from the co- occurrence matrix that can be used to represent the texture more compactly. Here are three numeric features that we used:

**Contrast(CON)**: This feature highlightsco-occurance of very different gray levels.

$$CON = \sum_{i=1}^{K} \sum_{j=1}^{K} (i-j)^2 P_d(i,j)/R$$

**Entropy(ENT)**: Entropy emphasises many different co-occurrences.

$$ENT = \sum_{i=1}^{K} \sum_{j=1}^{K} P[i,j] \ln \left( \frac{1}{P[i,j]} \right)$$

where $P[i,j]$ is the normalized co-occurrence matrix, each entry indicating probability of occurrence of that gray level combination.

**Inverse Difference Moment (IDM)**: IDM emphasises co-occurrence of close gray levels compared to highly different graylevels. **m** and **r** can user specified:

$$IDM = \sum_{i=1}^{K} \sum_{j=1}^{K} \frac{P_d^r[i,j]}{|(i-j)^m|_{i \neq j}}$$

### *E. Algorithm for image segmentation*

- Specify a window of size $w$ x $w$
- For the pixels in the window, compute the co- occurrence matrix
- Derive the texture features from the co-occurrence matrix
- Move the window by 1 pixel, and repeat the procedure

The procedure leads to texture images that may be treated like additional bands, equal to the number of features computed.

### *F. Classification*

The K-means clustering algorithm is employed to segregate pixel features by assigning them to distinct clusters based on their similarity. The user can input the desired number of clusters. Various techniques such as the Elbow method can be used to determine the optimal number of clusters. The output of the K-means algorithm is a set of clusters, where pixels with similar features are assigned to the same cluster.

## IV. GUI

The Graphical User Interface (GUI) has been designed in such a way that on running the executable it has options for selecting input image and the value for the number of clusters. On clicking ”process image”, users are required to type the output file name while the algorithm runs in the background. This has been made for easy usage instead of running the code multiple times.

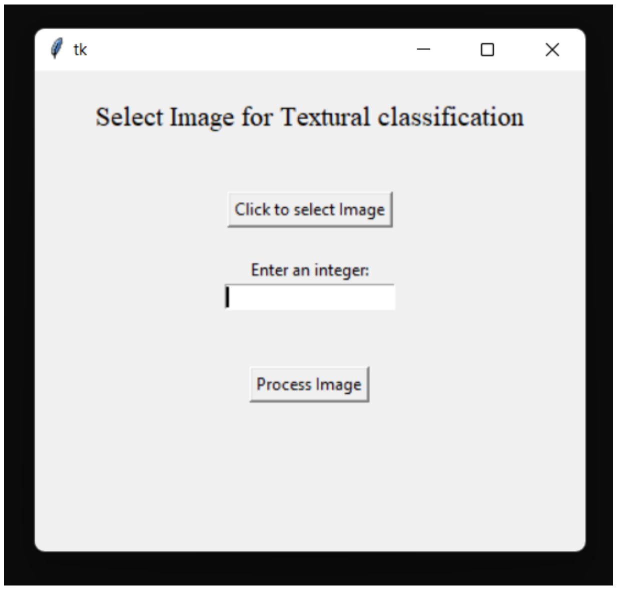

Fig. 2. Graphical User Interface

## V. Results

Some test images inputted to the executable with values for the number of clusters and the resultant images after processing are given below. Taking the Fig. 2. as an example we can see the different textures as being clearly classified. The entire water body is classified as a blue colour, the buildings around are classified as a different texture as the colour yellow and the land texture can be differentiated by the colour dark violet. Similarly in Fig. 3. and Fig. 4., the land and water textures have been classified as the resultant images show yellow, blue and green hues differentiating them.

## VI. Limitations

One of the major limitations of GLCM was its time-consuming nature which we have overcome by implementing fast GLCM while benefiting from its strength in extracting texture features. However, there are several limitations to this method of implementing fast GLCM. Firstly, it is computationally expensive. The whole image is divided into smaller images which require several matrices to be computed increasing

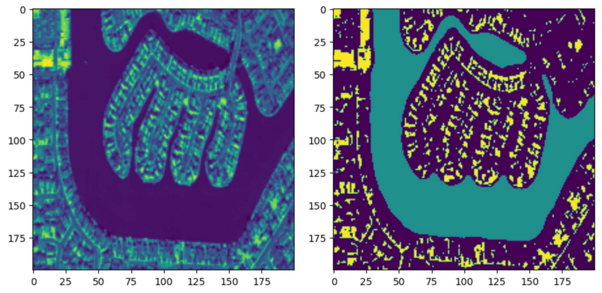

Fig. 3. Input (left) and Result (right) - 1

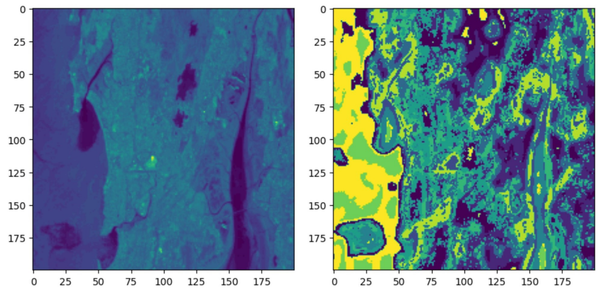

Fig. 4. Input (left) and Result (right) - 2

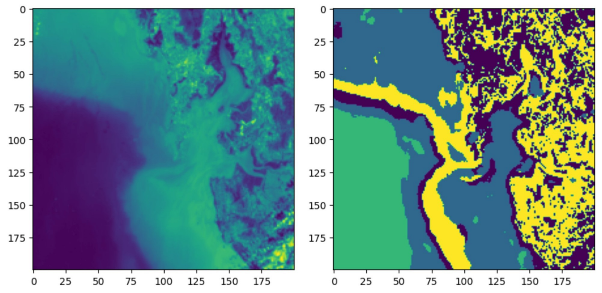

Fig. 5. Input (left) and Result (right) - 3

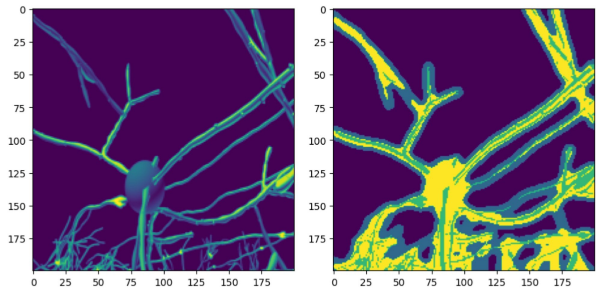

Fig. 6. Input (left) and Result (right) - 4

the overall number of computations. Another drawback is that we are not taking many textural features for processing. Only three numeric features: Contrast (CON), Entropy (ENT) and Inverse Difference Moment (IDM) are being used for processing. There are several other numeric features such as Angular Second Moment (ASM) etc which can also be computed to represent the texture more compactly but we have omitted them in our implementation. Furthermore, the number of clusters for K-means poses a limitation as we have taken a certain fixed number of labels depending on the input images such as for land, water etc but for an arbitrary image we don't know the number of labels required for proper classification.

## VII. Conclusion

This paper has presented a method for implementing Fast GLCM on remote sensing images. Firstly, we discussed the difference between GLCM and Fast GLCM and the advantages the former provides over the latter. Then we gave the methodology and approach to how we went onto writing the code from the algorithm of the construction of GLCM, numeric features from it, algorithm for image segmentation and finally classification. The easy to use GUI was presented along with the results to some input images were presented to show the classification. Finally, some limitations in our approach were discussed.

## References

[1] Mirzapour, Fardin & Ghassemian, Hassan. (2015). Fast GLCM and Gabor Filters for Texture Classification of Very High Resolution Remote Sensing Images. IJICTR, International Journal of Information Communication Technology Research. 7. 21-30.

[2] M. Tuceryan and A. K. Jain, ”Texture analysis,” Handbook of pattern recognition and computer vision, vol. 2, pp. 207-248, 1993.

[3] J. Zhang and T. Tan, ”Brief review of invariant texture analysis methods,” Pattern Recognition, vol. 35, pp. 735-747, 2002.